
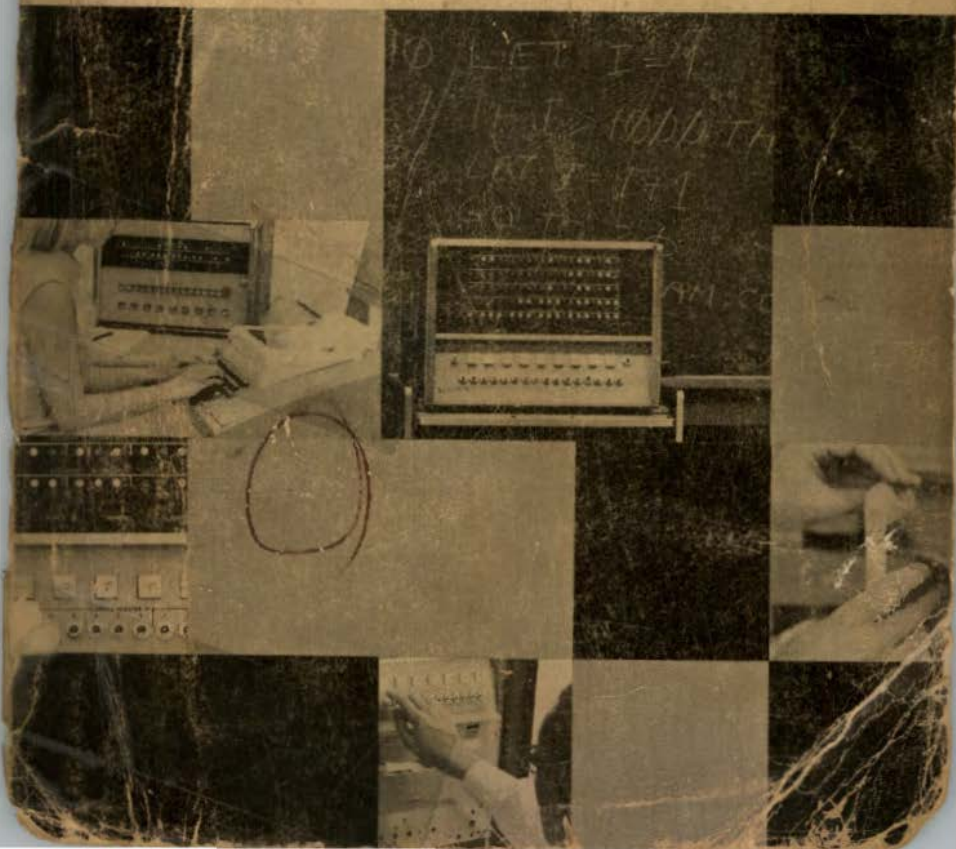


SCOTT MINNIN

2567 -- Math Dgt.
2561 GSA computer

HEWLETT  PACKARD

A Pocket Guide to Hewlett-Packard Computers



PREFACE

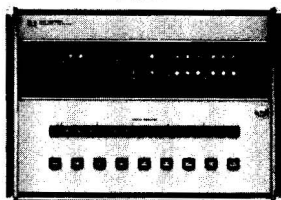
This manual combines in one convenient publication comprehensive hardware specifications for the Hewlett-Packard 2116B, 2115A and 2114A Computers and programmer's reference manuals for the principal software systems. Software manuals included are FORTRAN, BASIC, Assembler, Basic Control System and Program Library. System designers and programmers will find this book a handy, permanent reference. Potential users will find the technical descriptions valuable for evaluating Hewlett-Packard computers and supporting software.

Further information is available by calling your local Hewlett-Packard field office; one of more than 130 Sales and Service Offices throughout the world.

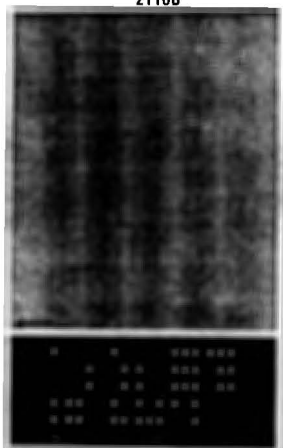
Or write Hewlett-Packard, 1501 Page Mill Road, Palo Alto, California 94304; Europe, 54 Route des Acacias, Geneva.

Here is a family of computers with the power to solve problems for engineers and scientists — at a cost that makes them uniquely practical.

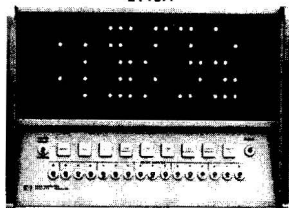
2114A



2116B



2115A



..... Backed up by software which is compatible to all three computers.

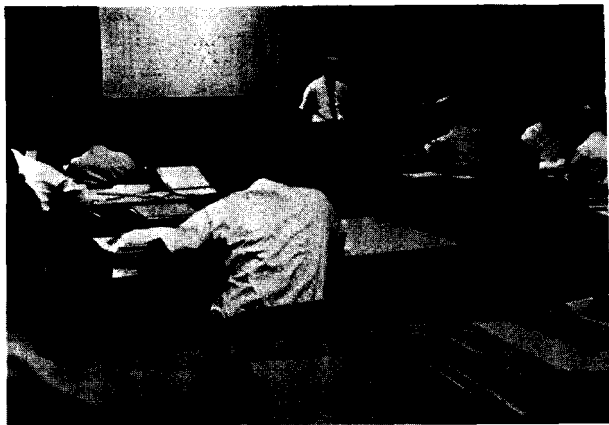
Hewlett-Packard computers combine performance and economy with small size. Achieved by simplicity of design — in package, in hardware, in software. A package that's easy to set up, with peripherals interfaced through plug-in cards. All modular for easy expansion. Straight forward machine organization and consoles that are easy to use. Integrated circuit construction that means long life, no quick obsolescence. A full-range of proven software packages — compatible between all models. All designed for busy engineers and scientists who want tomorrow's answers today.

FEATURES

- Low cost
- Proven software
- 16-bit word size
- 1.6 μ S memory cycle time
- Large 1024-word page size
- Powerful instruction set of 70 basic instructions
- Peripherals interfaced simply with plug-in cards
- Multilevel priority interrupt for device servicing
- Two accumulators, both addressable to simplify programming
- Extended Arithmetic capability available as plug-in option
- Magnetic core storage expandable to 32,768 words
- Protected loader
- Multiplexed I/O available
- Power Fail option preserves status, restarts automatically
- Two optional high-speed Direct Memory Access channels
- Modular I/O drivers — for device independent programming
- Two-pass ASA Basic FORTRAN — extended
- Modular Debug package — for on-line program debugging
- ALGOL and BASIC language compilers

LOW COST COMPUTERS WITH HIGH-PRICED PERFORMANCE

IN-DEPTH TRAINING IN PROGRAMMING AND MAINTENANCE

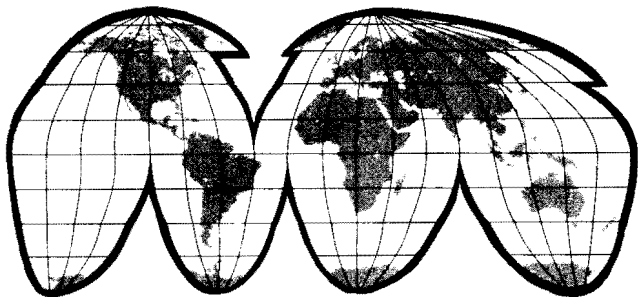


USER (PROGRAMMING) TRAINING

Hewlett-Packard provides a free user-programmer course for customers at the factory in Palo Alto, California. Training materials are also provided at no charge. The complete User Training Course assumes no knowledge of computer programming or electronic systems operation. It covers instruction on programming languages and operating system. At least two full days are devoted to hands-on experience.

MAINTENANCE TRAINING

Regularly scheduled Maintenance Training Courses for customers are also available at the factory in Palo Alto, California, at no charge for attendance or classroom supplies. The course assumes familiarity with digital logic circuits and covers the following subjects in depth: computer organization, logic operation and timing, I/O interfaces, fault diagnosis, and repair.



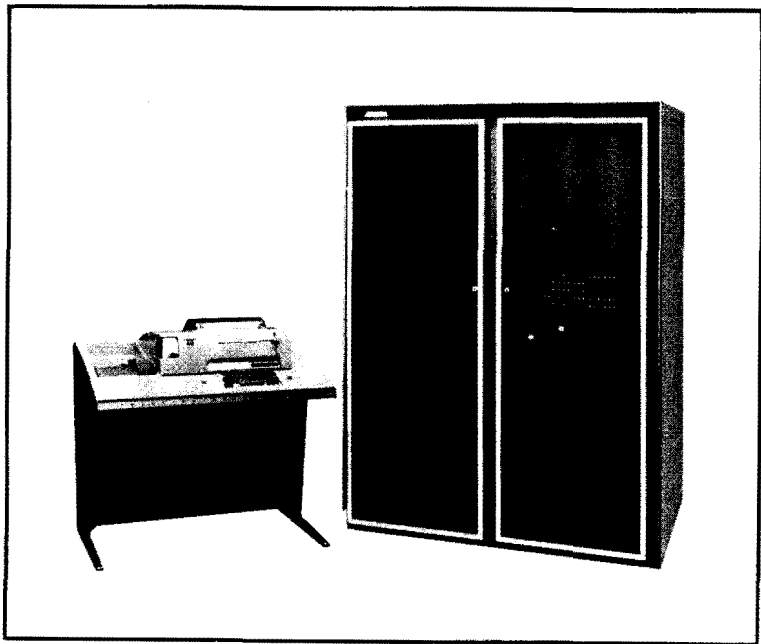
WORLD-WIDE REPAIR AND PARTS SERVICE

Service and parts assistance are available from Hewlett-Packard field offices throughout the United States, Canada and Europe. Local office facilities are backed up by Regional Service Centers. Major parts warehouses are located in Mountain View, California and Paramus, New Jersey. Parts orders are filled promptly; Hewlett-Packard uses a computer-controlled parts ordering and processing system which ensures that over 90% of orders for replacement parts are shipped the same day they are received.

WARRANTY

All Hewlett-Packard products are warranted against defects in materials and workmanship. This warranty applies for one year from date of delivery, or in the case of certain major components listed in the operating manual, for the specified period. We will repair or replace products which prove to be defective during the warranty period provided they are returned to Hewlett-Packard. No other warranty is expressed or implied. We are not liable for consequential damages.

Service contracts or customer assistance agreements are available for HP products that require maintenance and repair on-site.



Illustrative of HP 2116B Computer power is its use in the HP 2000A Time Sharing System. Here, up to 16 users may communicate with the computer simultaneously, in conversational BASIC language.

CONTENTS

Specifications and Basic Operation Manual

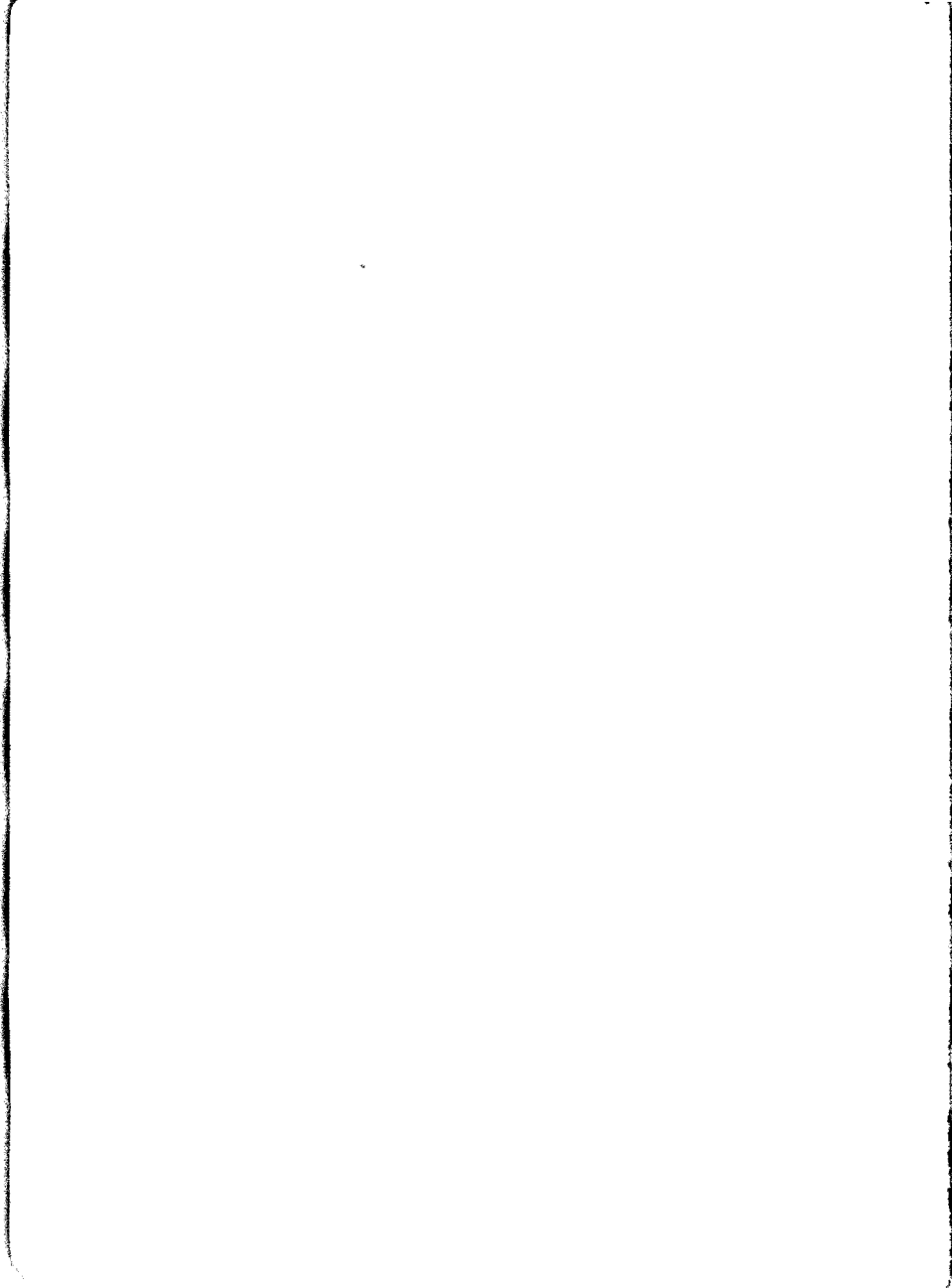
Assembler Reference Manual

Basic Control System Reference Manual

FORTRAN Reference Manual

Program Library Reference Manual

BASIC Language Reference Manual



Specifications and Basic Operation Manual



TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1	Interfacing	1-4
1.2	Input/Output Peripheral Devices	1-4
1.3	System Documentation	1-5

CHAPTER 2 COMPUTER SPECIFICATIONS AND DESCRIPTION

2.1	Options	2-1
2.2	Physical Specifications	2-1
2.3	Computer Timing	2-5
2.4	Memory	2-7
2.5	Working Registers	2-9
2.6	Computer Controls	2-11
2.7	Instructions	2-15
2.8	Data Formats	2-30
2.9	Input/Output Specifications	2-30
2.10	Processor Options	2-43
2.11	Input/Output Options	2-43
2.12	Software	2-51

CHAPTER 3 COMPUTER OPERATION

3.1	HP Computer Structure	3-1
3.2	Implementation of Instructions	3-18

ILLUSTRATIONS

Figure	Title	Page
1.1	Hewlett-Packard Computers	1-1
2.1	Dimensions of Computers	2-4

ILLUSTRATIONS(cont'd)

Figure	Title	Page
2.2	Computer Timing	2-6
2.3	Computer Panel Controls	2-12
2.4	Basic Instruction Formats	2-17
2.5	Memory Reference Instructions	2-19
2.6	Shift-Rotate Instructions	2-23
2.7	Alter-Skip Instructions	2-23
2.8	Input/Output Instructions	2-27
2.9	Basic Data Format	2-30
2.10	Input/Output Design Arrangement	2-35
2.11	Components of Typical Input/Output Interface Cards	2-36
2.12	Party Line Block Diagram	2-37
2.13	Multiplexed I/O Signal Lines	2-40
2.14	Input/Output Option Locations	2-48
3.1	Simplified Block Diagram for Computers	3-2
3.2	Memory Block Diagram	3-3
3.3	Core Memory Module	3-4
3.4	Binary Storage in a Magnetic Core	3-4
3.5	Core Addressing, Reading, and Writing	3-5
3.6	Memory Cell Selection	3-7
3.7	A Bit Plane and Frame (Upper Left Corner)	3-8
3.8	Register Block Diagram	3-9
3.9	Bus System Block Diagram	3-13
3.10	Instruction Logic Block Diagram	3-15
3.11	Input/Output System Block Diagram	3-17
3.12	Implementing Memory Reference Instructions	3-21
3.13	Implementing Register Reference Instructions	3-23
3.14	Implementing Input/Output Instructions	3-25
3.15	Register Manipulations for Indirect Jump	3-28
3.16	Register Manipulations for Indirect AND	3-30

TABLES

Table	Title	Page
1.1	Comparison of Features for Hewlett-Packard Digital Computers	1-2
2.1	Computer Instructions	2-16
2.2	Logic Truth Table	2-19
2.3	Consolidated Coding Table	2-32
2.4	Computer I/O Channels	2-34
2.5	Select Code Assignments	2-41
2.6	Input/Output Options	2-46
3.1	Shift-Rotate Functions	3-37

The Hewlett-Packard Models 2116B, 2115A, and 2114A Computers are small, general purpose computers which are particularly suited, in computational power and input/output flexibility, to educational, scientific, and industrial measurement applications. See Figure 1.1. The logical design and software follow conventional standards of computer usage and notation so that each HP computer may be used as a free-standing device or in systems, such as process control, media conversion, data reduction, communications systems, or time-sharing systems. The three computers vary in size and in memory and input/output capacity, but all use identical software and interface components. The capabilities of the three Hewlett-Packard computers are compared in Table 1.1.

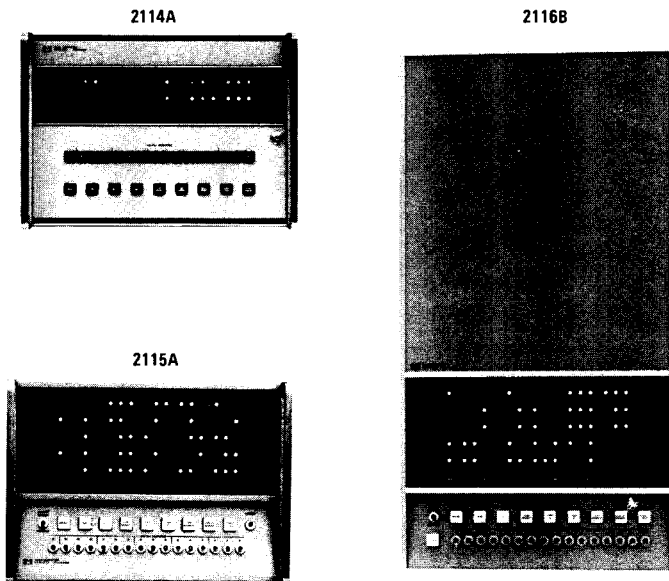


Figure 1.1. Hewlett-Packard Computers

Table 1.1. Comparison of Features for Hewlett-Packard Digital Computers

FEATURE	2116B	2115A	2114A
MEMORY SIZE			
Basic Configuration (16-bit words)	8,192	4,096	4,096
Maximum Available Memory	32,768	8,192	8,192
Expandable in Mainframe to:	16,384	8,192	8,192
WORD SIZE			
Data Bits	16	16	16
Memory Parity with Interrupt	Optional	Optional	Optional
Memory Protection (Area Protect)	Optional	No	No
MEMORY CYCLE (usec)	1.6	2.0	2.0
INSTRUCTION EXECUTION SPEED (usec)			
Store Word	3.2	4.0	4.0
Add (full word)	3.2	4.0	4.0
Multiply (subroutine max. time)	150	187	187
Divide (subroutine max. time)	310	387	387
Multiply (hardware) EAU Option	19.2	24.0	-
Divide (hardware) EAU Option	20.8	26.0	-
Number of Instructions	70	70	70
MULTILEVEL INDIRECT ADDRESSING	Yes	Yes	Yes
HIGH SPEED BUFFERED I/O CHANNELS			
Maximum number available (DMA Option)	2 (Assignable)	2 (Assignable)	0
Maximum word transfer rate (per second)	625,000	500,000	N/A
Cycles required to set-up block transfer	13	13	N/A
Cycles stolen (from main program) per word transferred	1	1	N/A

PRIORITY INTERRUPTS			
External interrupts in basic unit	16	8	8
Maximum number of external interrupts	48	40	8
Program enable/disable of individual interrupts	Yes	Yes	Yes
ENVIRONMENTAL TEMPERATURE, HUMIDITY	0°-55°C RH 95% at 40°C	10°-40°C RH 80% at 40°C	10°-40°C RH 80% at 40°C
CIRCUITRY	Integrated Circuits (CTL)	Integrated Circuits (CTL)	Integrated Circuits (CTL-TTL)
POWER FAILURE PROTECTION	Yes	Yes	Yes
Automatic Re-start	Optional	Optional	Optional
PARTY LINE	Yes	Yes	Yes
MULTIPLEXED I/O	No	No	Yes
COMPILER			
FORTRAN (Extended ASA Basic FORTRAN)	Yes	Yes	Yes
ALGOL (A subset of ALGOL 60)	Yes	Yes (8K memory)	Yes (8K memory)
BASIC	Yes	Yes (8K memory)	Yes (8K memory)
ASSEMBLER	Yes	Yes	Yes
REAL TIME EXECUTIVE SOFTWARE SYSTEM	Yes	No	No
TIME SHARED BASIC SYSTEM	Yes	No	No

1.1 INTERFACING

Interfacing an HP computer with a peripheral device consists of inserting one or two standard microcircuit cards in easily accessible input/output slots in the computer and connecting the device cable. Each HP computer provides a unique channel identification and service priority interrupt for every input/output channel used. Priority levels of the peripheral equipment connected to the computer can be altered simply by changing the positions of the interface cards in the I/O slots. Each computer operates with virtually all Hewlett-Packard measurement instruments providing a digital data output.

a. Digital voltmeters and associated signal converters, for measuring dc and ac voltages, currents, and resistances. With suitable transducers, physical quantities such as pressures, loads, temperatures, and fluid flows can be measured with an HP computer.

b. Electronic counters, for frequency or period measurements from a few cycles per second into the microwave region.

c. Scaler timers for nuclear radiation measurements.

d. Quartz thermometers for high-resolution temperature measurements in chemical analysis and oceanography.

Analog input scanners are available for multiplexing signals into these measuring instruments. Digital scanners are also available for applications where it is desirable to multiplex the data outputs of these instruments before entry into the computer. Table 2.6 in Chapter 2 provides further detail on input/output options.

Off-the-shelf interface cards enable the customer to operate a wide variety of devices of his own choosing with any HP computer. These include a 16-Bit Duplex Register card, a Relay Output card, a D-to-A Converter card, Party Line Input/Output (two cards) to enable computer users to interface many devices of their own choosing, and Multiplexed Input/Output (two cards) for connection of up to 56 devices to the 2114A Computer.

1.2 INPUT/OUTPUT DEVICES

Instructions or data may be entered on punched tape through a teleprinter or high-speed photoelectric tape reader. Data output devices include the teleprinter, which provides typewritten and punched tape records, a high-speed tape punch, magnetic tape units (for IBM-compatible, 7- and 9-channel recording) and line printers. A disc memory (32 or 64 track) is available for 2116B and 2115A

Computers. Data can be entered on-line from Hewlett-Packard data sources and computed in real time, or recorded on punched tape or magnetic tape and processed subsequently by the computer. For example, each computer accepts data from punched tapes and magnetic tapes recorded by HP data acquisition systems. A Data-Phone interface is also available which enables information to be transmitted over the telephone system, into or out of the HP computer.

1.3 SYSTEM DOCUMENTATION

Full documentation is provided with each computer or computer system shipped to a customer and consists of four volumes as follows:

a. Volume One: Specifications and Basic Operation Manual. This volume describes the full capabilities of the basic computer for which it is supplied (2116B, 2115A, or 2114A, as applicable), including standard hardware options, standard software, and fundamental computer operation. This manual is intended both as a reference for users who are familiar with computer terminology and as a source of detailed definitions, so the material will be meaningful to a wide range of readers.

b. Volume Two: Installation and Maintenance Manual. This volume gives installation and maintenance information for the basic processor unit only. Its contents include Installation, Performance Checks, Theory of Operation, Logic and Schematic Diagrams, and a Parts List.

c. Volume Three: Input/Output System Operation Manual. This volume describes the input/output structure and the standard Input/Output Options (referenced by Interface Kit numbers), which form the basis for all HP computer systems. Procedures are given for connecting, operating, and programming the input/output devices for which Hewlett-Packard designs interface cards. The information in these sections condenses operating procedures from the manuals of the individual instruments, and adds material relating specifically to operation with the applicable HP computer. Maintenance information in these sections covers only the interface circuits, and not the peripheral itself. Complete Operating and Service Manuals for the peripheral device are furnished when the device is included in a particular system.

d. Volume Four: Programmer's Reference Manuals. This volume consists of one or more 3-ring binders containing documentation for each item of software supplied with the computer. Both

standard software programs and software specially originated for an individual user are fully described as to specifications and usage. A "Software System Installation Record" at the front of Volume Four lists all software furnished with the original shipment, and provides an index to the supporting documents in Volume Four. Space is provided for noting changes and additions, so that an up-to-date record can be maintained by the user. Printed listings for furnished standard programs are supplied as manual supplements in this Volume. Programmer's Reference Manuals normally included in Volume Four are listed below. (This pocket manual includes the first five of the listed reference manuals.)

- 1) Assembler
- 2) BASIC Language
- 3) Basic Control System
- 4) FORTRAN
- 5) Program Library
- 6) Standard Software Systems Operating Manual
- 7) Symbolic Editor
- 8) ALGOL

In addition to the four volumes shipped with each computer, a Computer Training Manual is also available. The Training Manual is designed to provide a general introduction to digital computers, how small computers operate, and the concepts of programming.

This chapter provides specifications for the HP 2116B, 2115A, and 2114A Computers and describes them in terms of memory, working registers, programming instructions, input/output options, and software provided. All specifications apply to the basic computer unit only, unless specifically denoted as an Option specification. All information applies to all computers unless specified otherwise.

2.1 OPTIONS

Options for the HP computers are of two general types:

a. **Processor Options.** These options extend the memory and computation capabilities of the basic unit, and are identified by "M" numbers.

b. **Input/Output Options.** These options add input and/or output facilities to the basic HP computer. The option, identified by a "K" number (e. g., K1, K2), consists of an interface kit which provides the circuitry, cabling, and software to enable the computer to operate with a specific input or output instrument (measuring, reading, or recording device), or with a series of instruments. Compatible instruments, not included in the Interface Kit, are separately available from Hewlett-Packard. When external devices are connected to the computer, it then becomes part of a computer system.

2.2 PHYSICAL SPECIFICATIONS

2.2.1 POWER REQUIREMENTS

a. **2116B Line Voltage:** 115 Vac (15A) or 230 Vac (7.5A), changeable by internal jumpers. Voltage tolerance is $\pm 10\%$.

2115A Line Voltage: 115 Vac (11A) or 230 Vac (5.5A), changeable by internal jumpers. Voltage tolerance is $\pm 10\%$.

2114A Line Voltage: 115 Vac $\pm 10\%$ (7A). 230 Vac (3.5A) operation requires an external transformer.

b. **Line frequency** 50 to 60 Hz, $\pm 10\%$.

c. Main unit power consumption with internal supply loaded to capacity by plug-in options:

- 2116B: 1600 watts maximum.
1000 watts minimum (with Teleprinter Option).
- 2115A: 1200 watts maximum.
700 watts minimum (with Teleprinter Option).
- 2114A: 800 watts maximum.
500 watts minimum (with Teleprinter Option).

d. Power cable: Standard 3-prong connector (two power, one grounding).

2.2.2 ENVIRONMENTAL LIMITS

a. Temperature:

- 2116B: 0° to 55°C (32° to 131°F)
- 2115A/2114A: 10° to 40°C (50° to 104°F)

b. Relative Humidity:

- 2116B: to 95% at 40°C
- 2115A/2114A: to 80% at 40°C

2.2.3 VENTILATION

a. 2116B: Intake on sides and back at bottom, exhaust at top.

Air Flow: 600 cubic feet per minute.

Heat Dissipation: 5500 BTU/hour, maximum.

b. 2115A and 2114A: Intake at rear, exhaust on sides.

Air Flow: 400 cubic feet per minute.

2115A Heat Dissipation: 3500 BTU/hour,
maximum.

2114A Heat Dissipation: 2200 BTU/hour,
maximum.

2.2.4 PHYSICAL DIMENSIONS (See Figure 2.1)

a. Width: 16-3/4 inches, adapters are furnished for mounting in a standard 19-inch rack.

b. Panel Height:

- 2116B: 31-1/2 inches
- 2114A/2115A: 12 inches
- 2115A Power Supply: 10-1/4 inches

c. Depth:

2116B: 19-3/8 inches
2114A/2115A: 24-3/8 inches

d. Recommended cable clearance at rear: 5 inches minimum.

e. Recommended air exhaust clearance:

2116B: at top, 3 inches minimum.
2114A/2115A: at sides, 2 inches minimum.

f. Weight:

2116B: Net weight, 230 lb. (104 kg)
Shipping weight, 330 lb. (150 kg)
2115A: Net weight, 65 lb. (29, 5 kg)
Shipping weight, 99 lb. (44, 9 kg)
2161A Power Supply: Net weight, 95 lb. (43 kg)
Shipping weight, 138 lb. (62, 6 kg)
2114A: Net weight, 106 lb. (48 kg)
Shipping weight, 132 lb. (59, 9 kg)

2.2.5 SERVICE ACCESS

a. 2116B Computer:

1. Panel hinged at left edge (see dimensional illustration, Figure 2.1). Permits front access to input/output connectors, test switches, plug-in circuit boards, and panel wiring.

2. Main chassis slides forward out of cabinet and swings to right. Permits front access to back plane wiring, power supply, fuses, and 115/230V jumpers.

Note

Unstable mounting racks must not be used to mount the HP 2116B, due to weight shift forward when chassis is withdrawn for service. Table-top usage or Hewlett-Packard system cabinets are recommended.

b. 2115A and 2114A Computers:

1. Top panel slides back and up, permitting top access to input/output connectors, test switches, plug-in circuit boards, and wiring.

2. Bottom panel is removable for access to backplane wiring.

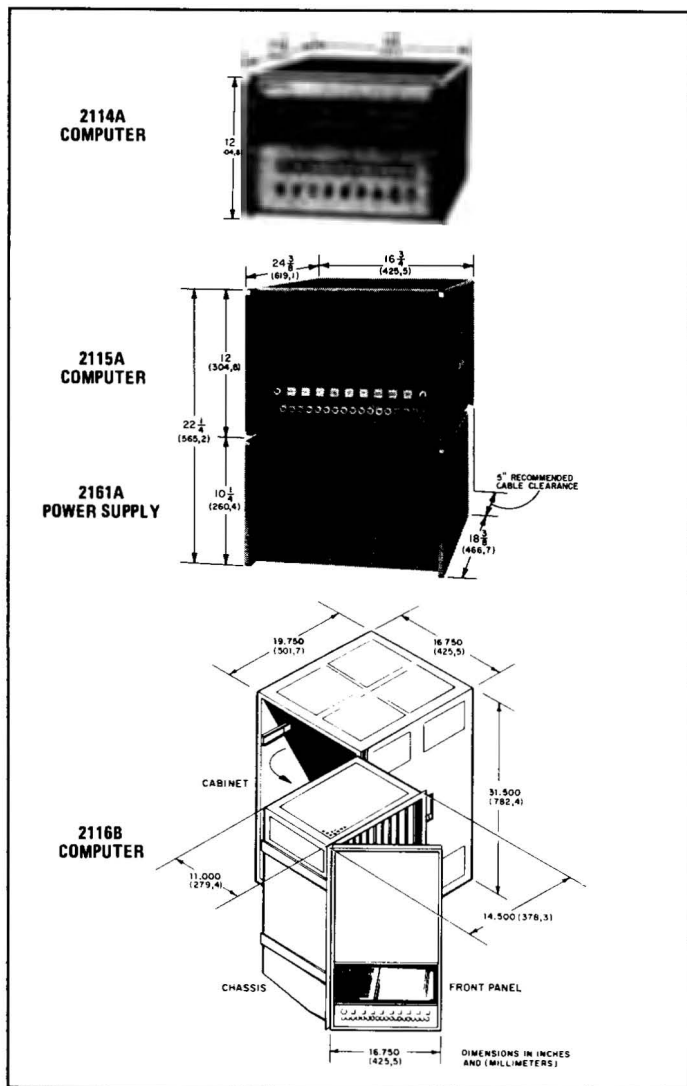


Figure 2.1. Dimensions of Computers

2.2.6 EXTENDER MODULES

The HP 2150A Input/Output Extender provides the HP 2116B and 2115A Computers with 32 extra channels of input/output capability.

The HP 2150B Input/Output Extender also provides 32 extra input/output channels for the 2116B and 2115A Computers. However, it allows the addition of 16,384 words of memory to the 2116B Computer, permitting an overall 2116B memory capability of 32,768 words.

Either extender may be ordered as an option. Each of the extenders is easily installed as a plug-in option with self-contained power supply.

2.3 COMPUTER TIMING

An internal 10-megahertz timing generator automatically generates read/write memory cycles every 1.6 microseconds in the 2116B Computer and every 2.0 microseconds in the 2115A and 2114A Computers. See Figure 2.2. Each HP computer has four machine phases (Fetch, Indirect, Execute, Interrupt), of which the first three include a memory cycle. Phases do not occur in a fixed sequence, but rather are determined by conditions which occur during operation. The computer can go directly from one of the first three phases to certain others in the manner indicated in Figure 2.2, and an external device can cause the computer to go into the Interrupt phase on completion of any current phase. The Fetch phase may be thought of as the "normal" or "home" condition; the processing of each instruction begins with a Fetch phase. Each phase of 2116B Computer operation takes 1.6 microseconds with one exception: the Execute phase of the ISZ instruction (Increment, and Skip if Zero) takes 2.0 microseconds. These times are 2.0 and 2.5 microseconds, respectively, for the 2114A and 2115A Computers.

FETCH PHASE. The contents of the currently-addressed memory cell is read into the T-Register during the Read portion of the memory cycle, and written back into the memory cell during the Write portion of the memory cycle. The information left in the T-Register is taken as an instruction when read during the Fetch phase. If the instruction includes an "indirect address bit", the computer sets the Indirect phase condition, and if the instruction does not have an indirect address bit but does include a memory reference (two-phase instruction), the computer sets the Execute phase condition. Otherwise the current instruction is fully executed at the end of the Fetch phase, and the computer remains in the Fetch state for the next memory cycle. An exception to these conditions is the JMP (jump) instruction, which is a Memory Reference instruction

but does not require an Execute phase; the computer executes the instruction at the end of the Fetch phase or the Indirect phase, and then sets the Fetch phase again for the next memory cycle.

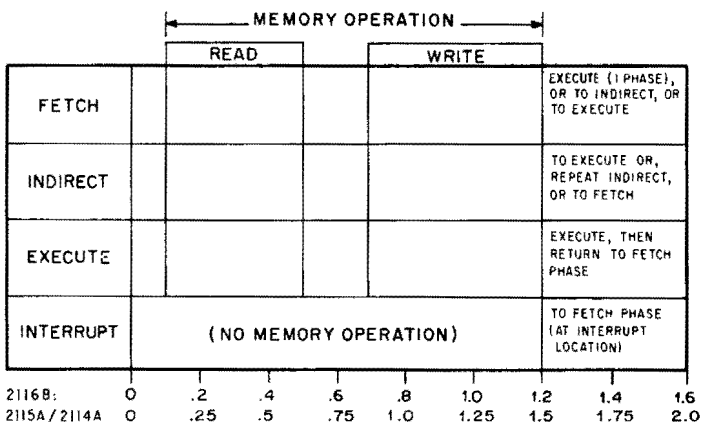


Figure 2.2. Computer Timing

INDIRECT PHASE. The contents of the memory cell referenced during the Fetch phase is read into the T-Register and the entire 16-bit word (15 bits of address, plus a new Direct/Indirect bit) is taken as a new memory reference for the same instruction. The use of 15 bits for an address permits addressing of up to 32,768 words. If the Direct/Indirect bit again specifies indirect addressing, the computer remains in the Indirect state and reads another 16-bit address word out of memory as a continuation of multiple-step indirect addressing. If the Direct/Indirect bit specifies direct addressing, the computer sets the Execute phase or, in the case of a Jump Indirect, the Fetch phase.

EXECUTE PHASE. The 16-bit data word in the memory cell referenced during a Fetch phase or an Indirect phase is read into the T-Register and is operated on by the current instruction (retained from the Fetch phase) at the end of the Execute phase. At the end of this phase, the computer sets the Fetch phase again to read the next instruction.

INTERRUPT PHASE. An input/output device requesting service at any time during one of the phases is acknowledged at the end of that phase, unless the interrupt is inhibited for any reason by the program being run. The computer then goes into the Interrupt phase,

which does not have a memory cycle. During this phase the P-Register is decremented, so that no instruction in the main program will be skipped or executed twice. At the end of this phase, the interrupt address of the interrupting device is transferred into the M-Register and the Fetch phase is set, to read the instruction contained in the interrupt address location. The Interrupt phase cannot occur again until at least this instruction is completed.

2.4 MEMORY

2.4.1 TYPE

Hewlett-Packard computer memories consist of ferrite-core storage modules. The 2116B Computer memory module is capable of storing 8192 words; the 2115A and 2114A module is capable of storing 4096 words. A word consists of 16 bits, plus a 17th bit when Memory Parity Option M2 is included in the computer. The 2116B permits the use of up to three additional modules to expand the storage capacity to 32,768 words. The 2115A and 2114A permit expansion to 8192 words.

2.4.2 LAYOUT

Each memory module is logically divided into pages of 1024 words each. A page is defined as the largest block of memory which can be addressed by the memory address bits of a Memory Reference instruction (excluding the Zero/Current page bit; see Figure 2.4). In HP computers, Memory Reference instructions have 10 bits to specify a memory address, and thus the page size is 1024 locations (2000 in octal notation). Octal addresses of the pages of the basic modules are therefore:

8192-word module or two 4096-word modules (8 pages)	4096-word module (4 pages)	00000 to 01777
		02000 to 03777
		04000 to 05777
		06000 to 07777
		10000 to 11777
		12000 to 13777
		14000 to 15777
		16000 to 17777

2.4.3 ADDRESSING

ZERO/CURRENT PAGE. For direct addressing purposes, generally only two pages are of interest: page Zero (the base page, consisting of locations 00000 through 01777), and the Current page (the page in which the instruction itself is located). All Memory Ref-

erence instructions include a bit (Bit 10) reserved to specify one or the other of these two pages. To address locations in any other page, indirect addressing is used. Page references for direct addressing of Memory Reference instructions are specified by Bit 10 as follows:

- 0 = Page Zero (Z)
- 1 = Current Page (C)

DIRECT/INDIRECT. All Memory Reference instructions use Bit 15 to specify direct or indirect addressing. Direct addressing combines the instruction code and the effective address into one word, permitting a Memory Reference instruction to be executed in two machine phases (Fetch and Execute). Indirect addressing uses the address part of the instruction word to access another word in memory which is taken as a new memory reference for the same instruction. This new address word is a full 16 bits long, 15 bits of address plus another Direct/Indirect bit. The 15-bit length of the address permits access to any location in any module. If Bit 15 again specified indirect addressing, still another address is obtained; this multiple-step indirect addressing may be done to any number of levels. The first address obtained in the Indirect phase which does not specify another indirect level becomes the effective address for the instruction. Instructions with indirect addresses are therefore executed in a minimum of three machine phases (Fetch, Indirect, Execute). Direct or indirect addressing is specified by Bit 15 as follows:

- 0 = Direct
- 1 = Indirect

RESERVED LOCATIONS. The first 64 memory locations of the base page (octal addresses 00000 through 00077) are reserved as listed below. The first two addresses are flip-flop registers (A and B accumulators) and not core storage locations. Locations 5 through 77 are reserved in the sense that interrupt wiring is present for the priority order given. As long as the locations do not have actual interrupt assignments (as determined by the input/output devices included in the user's system), these locations may be used for normal program purposes.

- | | |
|-------|--|
| 00000 | A Register |
| 00001 | B Register |
| 00002 | } For exit sequence if A and B contents
are used as executable words. |
| 00003 | |

00004	Interrupt location, highest priority. Reserved for Power Fail interrupt.
00005	Reserved for Memory Protect (2116B) and Memory Parity interrupt (2114A/ 2115A/2116B).
00006 } 00007 }	Reserved for DMA interrupt (2115A/ 2116B). Not used by 2114A.
00010 } thru } 00077 }	Interrupt locations in decreasing order of priority.

2.4.4 LOADER PROTECTION

The last 64 locations of memory are reserved for the Basic Binary Loader. The Basic Binary Loader is a manually-entered program to permit reading and storing of binary information from punched paper tape, as read by an HP 2737A Punched Tape Reader or an HP 2752A/2754A Teleprinter. Absolute addresses are required in the loaded data. On the 2115A and 2116B Computers, the front panel LOADER switch in the PROTECTED position disables the Basic Binary Loader locations so that they can neither be used nor altered in any way. For entering the Basic Binary Loader manually into the computer and for actual loading of tapes, this switch must be set to ENABLED. The LOADER switch is effective for the last 64 locations of memory, regardless of memory size. Plug-in options which expand memory relocate the protected area automatically to the 64 highest numbered locations. On the 2114A Computer, the same Basic Binary Loader locations are automatically protected during normal operation. For entering the Basic Binary Loader manually, the LOADER ENABLE switch, located behind the hinged, front panel, must be in the ON position.

2.5 WORKING REGISTERS

The HP 2116B, 2115A, and 2114A Computers have seven working registers. Five of these are 16-bit flip-flop registers, and two are 1-bit flip-flop registers. All seven registers are displayed on the front panel of 2115A and 2116B Computers. The 2114A Computer displays two 16-bit registers (Memory Data and Memory Address) and two 1-bit registers (Extend and Overflow). However, the contents of the 2114A A, B, and P registers can be displayed using these registers. The computer registers displayed are shown in Figure 2.3.

T-REGISTER (MEMORY DATA). All data transferred into or out of memory is routed through the 16-bit T-Register ("Transfer Reg-

ister"). The T-Register display therefore indicates exactly what information went into or out of a memory cell during the preceding memory cycle.

P-REGISTER (PROGRAM COUNTER). On completion of each instruction the P-Register indicates the address of the next instruction to be fetched out of memory. The P-Register automatically increments by one (or two, when executing a skip instruction) after the execution of each instruction. A jump instruction (JMP or JSB) can set the P-Register to any core location number. (On the 2114A, the P-Register will not increment if the HALT and LOAD MEMORY or HALT and DISPLAY MEMORY switches are touched simultaneously.)

M-REGISTER (MEMORY ADDRESS). The M-Register holds the address of the memory cell being read or written into. The M-Register indication will differ from the P-Register indication when multi-phase instructions are being processed, since the M-Register will be changed by memory references in the instruction (which may be several in the case of indirect addressing) or by an interrupt, whereas the P-Register remains constant until completion of the instruction.

A-REGISTER (ACCUMULATOR). The A-Register is an accumulator, holding the results of arithmetic and logical operations performed by programmed instructions. This register may be addressed by any Memory Reference instruction as location 00000, thus permitting inter-register operations such as "add B to A", "compare B with A", etc., using a single-word instruction.

B-REGISTER (ACCUMULATOR). The B-Register is a second accumulator, which can hold the results of arithmetic and logical operations completely independent of the A-Register. The B-Register may be addressed by any Memory Reference instruction as location 00001 for inter-register operations with A.

EXTEND. The Extend bit is a one-bit (E) register, and is used to link the A and B Registers by rotate instructions, or to indicate a carry from Bit 15 of the A or B Registers by an add instruction (ADA, ADB) or increment instruction (INA or INB, but not ISZ) which references these registers. This is of significance primarily for multiple-precision arithmetic. The Extend bit is not complemented by a carry if already set. It may be cleared, complemented, or tested by program instruction. The Extend bit is set when the EXTEND panel light is on ("1") and clear when off ("0").

OVERFLOW. The Overflow bit is a one-bit register which indicates, if on, that an add instruction (ADA, ADB) or an increment

instruction (INA or INB, but not ISZ) referencing the A or B registers has caused one of these accumulators to exceed the maximum positive or negative number which can be contained (+32767 or -32768, decimal). This condition is implied by a carry (or lack of carry) from Bit 14 to Bit 15. By program instructions, the Overflow bit may be cleared, set, or tested. The OVERFLOW panel light remains on until the bit is cleared by an instruction, and is not complemented if a second overflow occurs before being cleared. It will not be set by shift or rotate instructions.

2.6 COMPUTER CONTROLS

Panel controls for the HP 2114A, 2115A, and 2116B Computers are shown in Figure 2.3. The controls of the 2115A and 2116B are nearly identical; the only difference being the type and location of the POWER switch. The 2115A has a toggle-type ON/OFF switch on the right side of its front panel; the 2116B has a pushbutton ON/OFF switch on the left side of its front panel. Both have toggle switches for the Switch Register and all other control switches are pushbutton-type. The 2114A Computer contains only proximity-type sense switches on its front panel. Each switch requires only a touch for activation. The functions of the various switches are explained in the following paragraphs and apply to all computers unless otherwise specified.

SWITCH REGISTER. Each HP computer has a row of 16 switches on its front panel to enable manual entry of data into the computer. The 2115A and 2116B Computer switches are toggle-type; the up position is a "one", the down position is a "zero". The 2114A Computer switches are proximity-type sense switches which illuminate for a "one" and remain dark for a "zero". (On the 2114A, the CLEAR REGISTER switch resets all switches of the Switch Register to "zero".) The setting of the Switch Register may be entered in the computer in the following ways:

- a. By program, may be loaded into the A or B Register using LIA or LIB instructions with the Switch Register's Select Code (01).
- b. By program, may be merged (inclusive "or") into the A or B Register using MIA or MIB respectively.
- c. Manually, using LOAD ADDRESS switch, may be loaded into the P and M Registers (simultaneously), thus directing the computer to a specific memory cell.
- d. Manually, using LOAD MEMORY switch, may be entered into the memory cell specified by the M-Register (MEMORY ADDRESS), thus permitting the user to change the contents of any memory cell.

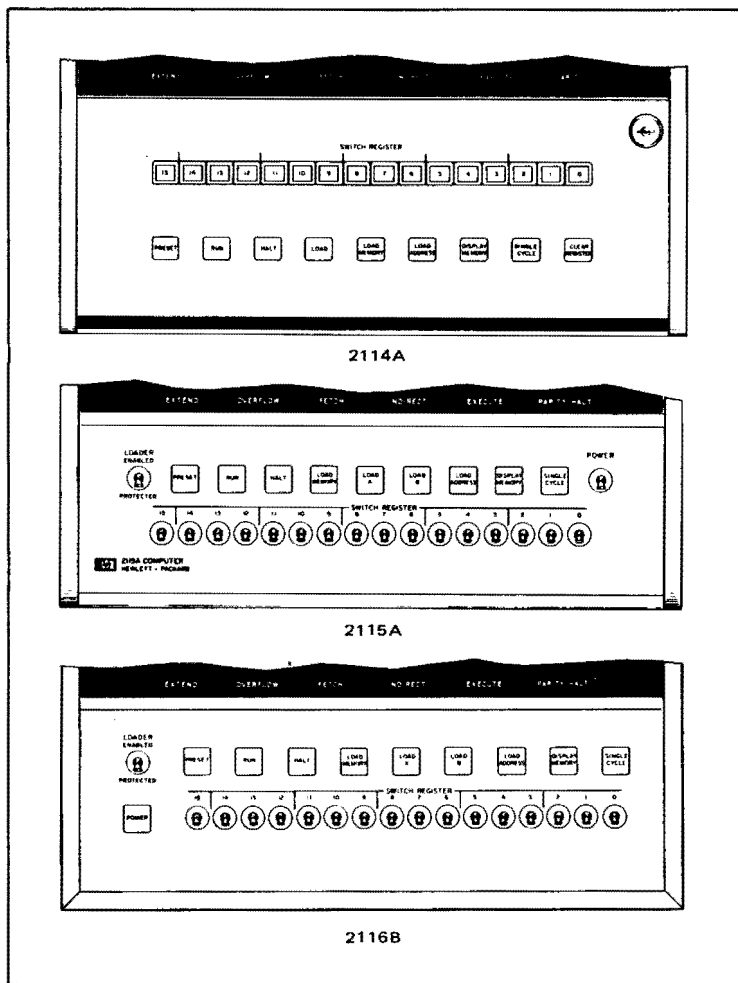


Figure 2.3. Computer Panel Controls

e. Manually, on a 2115A or 2116B Computer, using LOAD A or LOAD B switches, may be loaded into the A or B Registers.

Note

On the 2114A Computer only, the contents of the A or B Register may be output to the Switch Register and displayed by a programmed output instruction (OTA/B: Select Code 01). To display the A and B Register contents manually, press the CLEAR REGISTER switch, the LOAD ADDRESS switch, and then the DISPLAY MEMORY switch. The contents of the A-Register will be displayed in the MEMORY DATA register. The contents of the B-Register will be displayed in the MEMORY DATA register when the DISPLAY MEMORY switch is pressed again.

POWER. Regulated power to each of the HP computers automatically goes off in case of abnormal changes in internal power supplies. Contents of memory are not affected by switching power off and on; contents of working registers, however, are lost when power goes off (contents are random following turn on). The 2116B Computer POWER switch is a push-on/push-off switch which lights when regulated power is on. The 2114A and 2115A Computers have ON/OFF power switches; located on the right side of the 2115A front panel, and behind the front panel on the computer chassis of the 2114A (accessible only by using a key).

LOADER (2115A and 2116B)/LOAD (2114A). These switches are associated with the last 64 locations of memory; for example, octal addresses 07700 through 07777 in 4K computers, or 17700 through 17777 in 8K computers. These locations are normally occupied by the Basic Binary Loader.

a. With the LOADER toggle switch of the 2115A/2116B Computer in the ENABLED position, the last 64 locations of memory can be read or loaded; in the PROTECTED position, these locations are disabled.

b. The LOAD proximity-type switch on the 2114A is interlocked, electrically, with the PRESET switch. To load any absolute binary program using the last 64 locations of memory, these switches must be touched simultaneously.

PRESET. This switch presets the computer to the Fetch phase, to turn off the interrupt system and all input/output Control bits, and to set all input/output Flag bits. The switch resets the PARITY

HALT indication on the front panel of the 2116B and 2115A Computers, and the **PARITY** indication on the front of the 2114A. Also, it clears the power fail interrupt circuits. An internal preset pulse accomplishing the same functions is generated each time power is switched on or off.

RUN. This switch starts operation at the current state of the computer. The switch lights when a program is running, and goes off when **HALT** is pressed, when a **HLT** instruction is executed, or when an abnormal change occurs in the internal power supplies. When the **RUN** light is on, all front-panel control switches except **HALT**, **POWER**, and **LOADER** (2116B/2115A) are disabled, as well as the **CLEAR REGISTER** switch on the 2114A.

HALT. This switch stops computer operation at the end of the current phase. When the computer is halted, the **HALT** switch lights, and all front-panel control switches are enabled. (On the 2114A, the **P Register** will not increment if the **HALT** and **LOAD MEMORY** or **HALT** and **DISPLAY MEMORY** switches are touched simultaneously.)

LOAD MEMORY. This switch stores the contents of the Switch Register into the memory location specified by the address in the **M-Register**. The **P** and **M Registers** are automatically incremented after operation of the **LOAD MEMORY** switch, to simplify storing data into consecutive memory locations (exception for 2114A: see **HALT** switch description). The stored data remains displayed in the **T-Register** (**MEMORY DATA**), and the **Fetch** phase is set at the end of the load operation.

LOAD A (2115A/2116B only). This switch transfers the contents of the Switch Register into the **A-Register**. The computer's phase status is not altered.

LOAD B (2115A/2116B only). This switch transfers the contents of the Switch Register into the **B-Register**. The computer's phase status is not altered.

LOAD ADDRESS. This switch transfers the contents of the Switch Register into both the **P** and **M Registers**, thus directing the computer to the desired address. The **Fetch** phase is set at the end of the load operation.

DISPLAY MEMORY. This switch causes the display, in the **T-Register** (**MEMORY DATA**), of the contents of the location specified by the address in the **M-Register**. The **P** and **M Registers** are automatically incremented after operation of the **DISPLAY MEMORY** switch, so that consecutive memory locations may be dis-

played by repeated operation of this switch. The P and M Registers are therefore one step ahead of the T-Register display. The Fetch phase is set after incrementing of the P and M Registers. (Exception for 2114A; see HALT switch description.)

SINGLE CYCLE. This switch executes one machine phase each time it is pressed.

CLEAR REGISTER (2114A only). This switch resets the Switch Register to "zero".

LOADER ENABLE ON/NORMAL (2114A only). This switch is located on the inside of the front panel. The NORMAL position protects the contents of the last 64 locations in memory normally occupied by the Basic Binary Loader. In the ON position, the Loader is enabled to allow words in Loader area to be changed, the new Loader loaded in, etc.

LAMP TEST TEST/NORMAL (2114A only). This switch is located on the inside of the front panel. It permits testing of all indicator lights in the front panel.

CONSOLE LOCK LOCK/NORMAL (2114A only). This switch is located on the inside of the front panel. In the LOCK position, the front panel switches are disabled to permit continuous computer operation without accidental interruption or tampering. Wire jumpers on the switch printed-circuit card can be removed to permit the Switch Register switches or the bottom row of control switches to be disabled individually if desired. With the switch in the NORMAL position, all front panel switches are enabled.

Diagnostic Switches. Each HP computer has three switches on the inside of its front panel to aid the computer technician in diagnosing malfunctions in the basic unit. Switch nomenclature defines the functions performed by each switch: looping of a machine phase, looping of a single instruction, and turning memory on or off.

2.7 INSTRUCTIONS

2.7.1 NUMBER

The HP computers have 70 basic one-word instructions, all executable in 1.6 or 3.2 microseconds in the 2116B and 2.0 or 4.0 microseconds in the 2114A and 2115A, except for the ISZ instruction. The ISZ instruction is executable in 3.6 microseconds in the 2116B and in 4.5 microseconds in the 2114A and 2115A. These instructions are grouped in three types and are summarized in Table 2.1.

Table 2.1. Computer Instructions

TYPE	MNEMONIC	DESCRIPTION	2116B μSEC	2115A/2114A μSEC
Memory Reference (14 total)	AND	"And" (M) to A; result in A	3.2	4.0
	XOR	"Exclusive or" (M) to A; result in A	3.2	4.0
	IOR	"Inclusive or" (M) to A; result in A	3.2	4.0
	JSB	Jump to subroutine, save P	3.2	4.0
	JMP	Jump, unconditionally	1.6	2.0
	ISZ	Increment (M); skip if result zero	3.6	4.5
	ADA/B	Add (M) to A or B; result in A or B	3.2	4.0
	CPA/B	Compare (M) with A or B; skip if unequal	3.2	4.0
Register Reference (43 total)	LDA/B	Load (M) into A or B	3.2	4.0
	STA/B	Store A or B into M; A/B unchanged	3.2	4.0
	NOP	SHIFT-ROTATE GROUP No operation	1.6	2.0
	CLE	Clear E (Extend)		
	SLA/B	Skip if least significant bit of A/B is zero		
	A/BLS	A/B arithmetic left shift one bit		
	A/BRs	A/B arithmetic right shift one bit		
	RA/BL	Rotate A/B left one bit		
	RA/BR	Rotate A/B right one bit		
	A/BLR	A/B left shift one bit, sign cleared		
	ERA/B	Rotate E right one bit with A or B		
	ELA/B	Rotate E left one bit with A or B		
	A/BLF	Rotate A or B left four bits		
	CLA/B	ALTER-SKIP GROUP Clear A or B	1.6	2.0
	CMA/B	Complement A/B (ones complement)		
	CCA/B	Clear-complement A/B (set to -1)		
	CLE	Clear E (Extend)		
	CME	Complement E		
	CCE	Clear-complement E (set E)		
	SEZ	Skip if E is zero		
	SSA/B	Skip if sign of A/B is zero (positive)		
	SLA/B	Skip if least significant bit of A/B is zero		
	INA/B	Increment A/B by one		
	SZA/B	Skip if A/B is zero		
	RSS	Reverse skip sense		
Input/ Output (13 total)	STO	OVERFLOW Set overflow bit	1.6	2.0
	CLO	Clear overflow bit		
	SOC	Skip if overflow bit clear		
	SOS	Skip if overflow bit set		
	HLT	Halt program		
	STF	Set flag bit of selected I/O channel		
	CLF	Clear flag of selected I/O channel		
	SFC	Skip if flag clear		
	SFS	Skip if flag set		
	MIA/B	Merge ("or") I/O channel into A/B		
	LIA/B	Load I/O channel into A/B		
	OTA/B	Output A/B to I/O channel		
	STC	Set control bit of selected channel		
	CLC	Clear control bit of selected channel		

1. (M) = Contents of Memory Location M

2. Overflow instructions are coded under I/O group

Each of the instructions shown in Table 2.1 are further explained in the following paragraphs. Register Reference instructions can be combined to form additional one-word "microinstructions", executable in a single phase.

2.7.2 FORMATS

The three types of basic instructions are grouped according to the bit format of the instruction word. These types are: Memory Reference, Register Reference, and Input/Output instructions. A comparison of the three formats is given in Figure 2.4, and detailed binary coding is included with the instruction descriptions following.

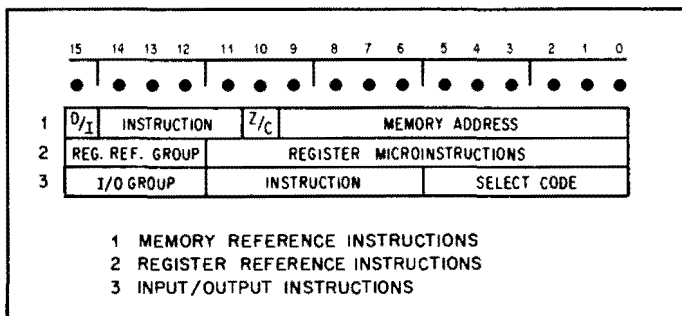


Figure 2.4. Basic Instruction Formats

The first type comprises the Memory Reference instructions, using 10 bits (0 through 9) for a memory address, Bit 10 to specify Zero or Current page, and Bit 15 for direct or indirect addressing. This leaves four bits (14, 13, 12, 11) to encode the 14 instruction commands in this group.

The other two types use four bits (15, 14, 13, 12) to distinguish the Register Reference and the Input/Output instructions. The Register Reference type uses Bits 11 through 0 to combine up to eight "microinstructions" (i.e., instructions formed by only 1, 2, or 3 bits), with the resulting multiple instruction operating on the A, B, or E Registers as a single-word instruction. The Input/Output type uses Bits 11 through 6 for a variety of input/output instructions, and Bits 5 through 0 to make the instruction apply directly to one of 64 possible input/output devices or functions.

The following paragraphs describe in detail each of the instructions in the three type groups. Functions of bits appearing in the form A/B, D/I, D/E, Z/C, or H/C throughout these Specifications are

invariably obtained by coding a 0 or 1 respectively (0/1). Thus, for example, A is specified by a zero-bit, and B by a one-bit. The following defines the abbreviations used:

A/B	A Register/B Register
D/I	Direct/Indirect
D/E	Disable/Enable
Z/C	Zero page/Current page
H/C	Hold Flag/Clear Flag

2.7.3 MEMORY REFERENCE INSTRUCTIONS

The 14 Memory Reference instructions execute some operation involving memory locations, such as transferring information in or out of a memory cell or checking the memory cell contents. The cell referenced (i.e., the absolute address) is determined by a combination of the ten memory address bits in the instruction word (0 through 9) and five bits (10 through 14) assumed from the current indication of the P-Register. This means that Memory Reference instructions can directly address any word in the current page; additionally, if the instruction is given in some location other than the base page (page Zero), Bit 10 of the instruction word doubles the addressing range to 2048 words by allowing selection of either page Zero or Current page (i.e., Bits 10 through 14 of the address in the M-Register can be reset to zero, instead of assuming the current indication of the P-Register). This feature provides a convenient linkage between all pages of memory, since page Zero can be reached directly from any other page.

Note that since the A and B Registers can be addressed, any Memory Reference instruction can apply to either of these registers as well as to memory cells. For example, ADA 0001 means add the contents of the B-Register (its address being 0001) to the A-Register; specify page Zero for these operations, since the A and B Register addresses are on page Zero.

Figure 2.5 gives instruction codes and mnemonics for all 14 Memory Reference instructions. All Memory Reference instructions take a minimum of two machine phases (one to read the instruction word, and one to read the referenced memory cell), except for JMP, which is a one-phase instruction. Logic truth tables, relating to the first three instructions described below, are given in Table 2.2. Note that logic operations are performed on a bit-for-bit basis (i.e., no carries).

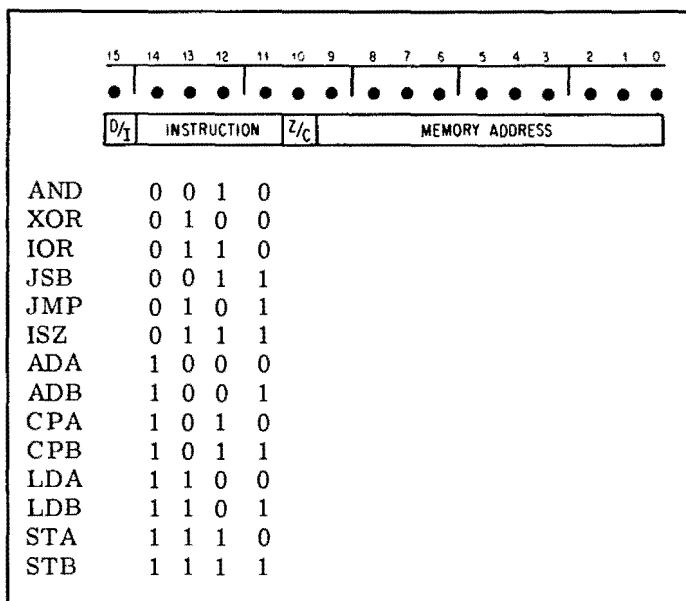


Figure 2.5. Memory Reference Instructions

Table 2.2. Logic Truth Table

	AND	XOR	IOR
A Contents	0 0 1 1	0 0 1 1	0 0 1 1
Memory	0 1 0 1	0 1 0 1	0 1 0 1
Result (in A)	0 0 0 1	0 1 1 0	0 1 1 1
1 = True, 0 = False			

AND. "And" to A. The contents of the addressed location are logically "anded" to the contents of the A-Register. The contents of the memory cell are left unaltered.

XOR. "Exclusive or" to A. The contents of the addressed location are combined with the contents of the A-Register as an "exclusive or" logic operation. The contents of the memory cell are left unaltered.

IOR. "Inclusive or" to A. The contents of the addressed location are combined with the contents of the A-Register as an "inclusive or" logic operation. The contents of the memory cell are left unaltered.

JSB. Jump to Subroutine. This instruction, executed in location P, causes computer control to jump unconditionally to the memory location (X) specified in the address portion of the JSB instruction word. The contents of the P-Register plus one (return address) is stored in location X, and the next instruction to be executed will be that contained in the next location (X + 1). A return to the main program sequence at P + 1 may be effected by a jump indirect through location X.

JMP. Jump. This instruction transfers control to the contents of the addressed location. That is, JMP causes the P and M Registers to be set according to the memory address portion of the instruction word, thus addressing memory cell X, so that the next instruction will be read from location X.

ISZ. Increment, and Skip if Zero. An ISZ instruction adds one to the contents of the addressed memory location. If the result of this operation is zero, the next instruction is skipped, i.e., the P and M Registers are advanced by two instead of one. Otherwise, the program proceeds normally to the next instruction in sequence. The incremented value is written back into the memory cell in either case. An ISZ instruction referencing locations zero or one (A or B Registers) cannot cause setting of the Extend or Overflow bits (unlike INA and INB).

ADA. Add to A. The contents of the addressed memory location are added to the contents of the A-Register, and the sum remains in the A-Register. The result of the addition may set the Extend or Overflow bits. The contents of the memory cell are unaltered.

ADB. Add to B. The contents of the addressed memory location are added to the contents of the B-Register, and the sum remains in the B-Register. Extend or Overflow bits may be set, as for ADA. The contents of the memory cell are unaltered.

CPA. Compare to A, skip if unequal. The contents of the addressed location are compared with the contents of the A-Register. If the two 16-bit words are different, the next instruction is skipped; i.e., the P and M Registers are advanced by two instead of one. If the words are identical, the program proceeds normally to the next instruction in sequence. The contents of neither the A-Register nor the memory cells are altered.

CPB. Compare to B, and skip if unequal. Same as CPA, except comparison is made with the B-Register.

LDA. Load into A. The A-Register is cleared and loaded with the contents of the addressed location. The contents of the memory cell are unaltered.

LDB. Load into B. The B-Register is cleared and loaded with the contents of the addressed location. The contents of the memory cell are unaltered.

STA. Store A. The contents of the A-Register are stored in the addressed location. The previous contents of the memory cell are lost; the A-Register is unaltered.

STB. Store B. The contents of the B-Register are stored in the addressed location. The previous contents of the memory cell are lost; the B-Register is unaltered.

2.7.4 REGISTER REFERENCE INSTRUCTIONS

The Register Reference instructions, in general, manipulate bits in the A, B, and E Registers. There is no reference to memory; thus these instructions are executed in only one machine phase. This type includes 39 basic instructions (plus four Overflow instructions, which are coded under the I/O Group instructions). Register Reference instructions are combinable to form a one-word multiple instruction that can operate in various ways on the contents of the A, B, or E Registers. These "microinstructions" are divided into two sub-groups, the Shift-Rotate Group (SRG) and the Alter-Skip Group (ASG). Three instructions (SLA, SLB, and CLE) appear in both groups and, being combinable in these different contexts, are counted twice in the total of basic instructions. Microinstructions may be combined under the following general rules:

- a. Instructions from the two groups cannot be mixed.
- b. References to both A and B Registers cannot be mixed.
- c. Only one microinstruction can be chosen from each column of the Selection Tables in Figures 2.6 and 2.7.
- d. Use zeros to exclude unwanted microinstruction bits.
- e. The sequence of execution is left to right in the Selection Tables (column 1, then column 2, etc.).
- f. If two (or more) skip functions are combined, the skip will occur if either or both conditions are met. One exception exists: see RSS under the Alter-Skip Group.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	TYPE 2				A/B	0	D/E	COL. 1				2	D/E	3	COL. 4	
NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SLA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SLB	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
ALS	0	0	0	0	0	0	1	0	0	0	0	X	X	X	X	X
BLS	0	0	0	0	1	0	1	0	0	0	0	X	X	X	X	X
ARS	0	0	0	0	0	0	1	0	0	0	1	X	X	X	X	X
BRS	0	0	0	0	1	0	1	0	0	0	1	X	X	X	X	X
RAL	0	0	0	0	0	0	1	0	1	0	1	X	X	X	X	X
RBL	0	0	0	0	1	0	1	0	1	0	1	X	X	X	X	X
RAR	0	0	0	0	0	0	1	0	1	1	1	X	X	X	X	X
RBR	0	0	0	0	1	0	1	0	1	1	1	X	X	X	X	X
ALR	0	0	0	0	0	0	1	1	0	0	0	X	X	X	X	X
BLR	0	0	0	0	1	0	1	1	0	0	0	X	X	X	X	X
ERA	0	0	0	0	0	0	1	1	0	1	1	X	X	X	X	X
ERB	0	0	0	0	1	0	1	1	0	1	1	X	X	X	X	X
ELA	0	0	0	0	0	0	1	1	1	1	0	X	X	X	X	X
ELB	0	0	0	0	1	0	1	1	1	1	0	X	X	X	X	X
ALF	0	0	0	0	0	0	1	1	1	1	1	X	X	X	X	X
BLF	0	0	0	0	1	0	1	1	1	1	1	X	X	X	X	X

SELECTION TABLE

1	2	3	4
ALS ARS RAL			ALS ARS RAL

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	TYPE 2				A/B	1	COL. 1		COL. 3		2	4	5	6	7	8
CLA	0	0	0	0	0	1	0	1								
CLB	0	0	0	0	1	1	0	1								
CMA	0	0	0	0	0	1	1	0								
CMB	0	0	0	0	1	1	1	0								
CCA	0	0	0	0	0	1	1	1								
CCB	0	0	0	0	1	1	1	1								
SEZ	0	0	0	0	1											
CLE	0	0	0	0	1					0	1					
CME	0	0	0	0	1					1	0					
CCE	0	0	0	0	1					1	1					
SSA	0	0	0	0	0	1										
SSB	0	0	0	0	1	1										
SLA	0	0	0	0	0	1										
SLB	0	0	0	0	1	1										
INA	0	0	0	0	0	1										
INB	0	0	0	0	1	1										
SZA	0	0	0	0	0	1										
SZB	0	0	0	0	1	1										
RSS	0	0	0	0	1											

SELECTION TABLE

1	2	3	4	5	6	7	8
CLA CMA CCA	SEZ	CLE CME CCE	SSA	SLA	INA	SZA	RSS

RAR ALR ERA ELA ALF	CLE	SLA	RAR ALR ERA ELA ALF
BLS BRS RBL RBR BLR ERB ELB BLF	CLE	SLB	BLS BRS RBL RBR BLR ERB ELB BLF
COMBINING GUIDE 1. Choose up to 4 instructions, one from each column of the Selection Table. 2. Use a one-bit for Bit 9 to Enable column 1 instructions, and a one-bit for Bit 4 to Enable column 4 instructions. Figure above shows column 1 enabled (executed first) with duplicate column 4 pattern (executed last) indicated by X's. 3. Use a one-bit for Bit 5 to select column 2 (CLE), or a zero-bit to exclude CLE. 4. Use a one-bit for Bit 3 to select column 3 (SLA/B), or a zero-bit to exclude SLA/B.			

Figure 2.6. Shift-Rotate Instructions

CLB CMB CCB	SEZ	CLE CME CCE	SSB	SLB	INB	SZB	RSS
COMBINING GUIDE 1. Choose up to 8 instructions, one from each column of the Selection Table. 2. Use the specified two-bit combinations of Bits 9 and 8, plus A/B Bit 11, to encode column 1 instructions. 3. Use the specified two-bit combinations of Bits 7 and 6 to encode column 3 instructions. 4. Use a one-bit in Bits 5, 4, 3, 2, 1, plus A/B Bit 11, to encode column 2, 4, 5, 6, 7 instructions respectively. 5. Use a one-bit for Bit 0 to encode column 8.							

Figure 2.7. Alter-Skip Instructions

Register Reference Instructions are recognized by the computer when the four most significant bits of the instruction word are zeros; the general format for this type of instruction (the dots representing variable microinstruction bits) is therefore:

0 000

SHIFT-ROTATE GROUP. The SRG instructions are specified by a zero for Bit 10 (compare Figures 2.6 and 2.7). Figure 2.6 gives both the bit format and the Selection Table for using these instructions. Definitions for the mnemonics used are listed below. Note that the Extend bit is not affected by shifts or rotates unless specifically stated. All of the shift and rotate instructions can be executed either first or last in a combined instruction, or both times. This permits sequencing of CLE and SLA/B either before or after shifts and rotates.

NOP	No Operation. Memory cycle only.
CLE	Clear E-Register.
SLA	Skip next instruction if Least Significant bit of A-Register is zero (i.e., skip if an even number is in A).
SLB	Skip next instruction if Least significant bit of B-Register is zero (i.e., skip if an even number is in B).
ALS	A-Register Left Shift one place, arithmetically (15 bits only). A zero replaces vacated Bit 0; bit shifted out of Bit 14 is lost; Bit 15 (sign bit) is not affected.
BLS	B-Register Left Shift one place, arithmetically (15 bits only). A zero replaces vacated Bit 0; bit shifted out of Bit 14 is lost; Bit 15 (sign bit) is not affected.
ARS	A-Register Right Shift one place, arithmetically. Bit shifted out of Bit 0 is lost; copy of sign bit (Bit 15) shifted into Bit 14; Bit 15 is not affected.
BRS	B-Register Right Shift one place, arithmetically. Bit shifted out of Bit 0 is lost; copy of sign bit (Bit 15) shifted into Bit 14; Bit 15 is not affected.
RAL	Rotate A-Register Left one place, all 16 bits. Bit 15 is rotated around to Bit 0.
RBL	Rotate B-Register Left one place, all 16 bits. Bit 15 is rotated around to Bit 0.
RAR	Rotate A-Register Right one place, all 16 bits. Bit 0 is rotated around to Bit 15.
RBR	Rotate B-Register Right one place, all 16 bits. Bit 0 is rotated around to Bit 15.

ALR	A-Register Left shift one place, same as ALS, but clear sign bit after shift.
BLR	B-Register Left shift one place, same as BLS, but clear sign bit after shift.
ERA	Rotate E-Register Right with A-Register, one place (17 bits). Bit 0 is rotated into Extend Register; Extend content is rotated into Bit 15.
ERB	Rotate E-Register Right with B-Register, one place (17 bits). Bit 0 is rotated into Extend Register; Extend content is rotated into Bit 15.
ELA	Rotate E-Register Left with A-Register, one place (17 bits). Bit 15 is rotated into Extend Register; Extend content is rotated into Bit 0.
ELB	Rotate E-Register Left with B-Register, one place (17 bits). Bit 15 is rotated into Extend Register; Extend content is rotated into Bit 0.
ALF	Rotate A-Register Left Four places, all 16 bits. Bits 15, 14, 13, 12 are rotated around to Bits 3, 2, 1, 0 respectively. Equivalent to four successive RAL instructions.
BLF	Rotate B-Register Left Four places, all 16 bits. Bits 15, 14, 13, 12 are rotated around to Bits 3, 2, 1, 0 respectively. Equivalent to four successive RBL instructions.

ALTER-SKIP GROUP. The ASG instructions are specified by a one in Bit 10. Figure 2.7 gives both the bit format and the Selection Table for using these instructions. Definitions for the mnemonics used are as follows:

CLA	Clear A-Register.
CLB	Clear B-Register.
CMA	Complement A-Register. One's complement, reversing the state of all 16 bits.
CMB	Complement B-Register. Reverses state of all 16 bits.
CCA	Clear, then Complement A-Register. Puts 16 ones in the A-Register; this is the two's complement form of -1.
CCB	Clear, then Complement B-Register. Puts 16 ones in the B-Register; this is the two's complement form of -1.

CLE	Clear E-Register.
CME	Complement E-Register. Reverses state of the Extend bit.
CCE	Clear, then Complement E-Register. Sets the Extend bit.
SEZ	Skip the next instruction if E-Register is zero.
SSA	Skip next instruction if Sign bit (Bit 15) of A-Register is zero; i.e., skip if the content of A is positive.
SSB	Skip next instruction if Sign bit (Bit 15) of B-Register is zero; i.e., skip if the content of B is positive.
SLA	Skip next instruction if Least significant bit of A-Register is zero (i.e., skip if an even number is in A).
SLB	Skip next instruction if Least significant bit of B-Register is zero (i.e., skip if an even number is in B).
INA	Increment A-Register by one. Can cause setting of Extend or Overflow bits.
INB	Increment B-Register by one. Can cause setting of Extend or Overflow bits.
SZA	Skip next instruction if A-Register is Zero (16 zeros).
SZB	Skip next instruction if B-Register is Zero (16 zeros).
RSS	Reverse Skip Sense. Skip occurs for any of the preceding skip instructions, if present, when the non-zero condition is met. RSS without a skip instruction in the word causes an unconditional skip. If a word with RSS also includes both SSA/B and SLA/B, both bits (Bit 15 and Bit 0) must be one for skip to occur; in all other cases the skip occurs if either or both conditions are met.

2.7.5 INPUT/OUTPUT INSTRUCTIONS

The HP computers have 17 basic Input/Output instructions, which provide the following general capabilities. (Note: Overflow instructions are associated with register reference operations and are therefore listed under Register Reference instructions in Table 2.1; however, they are coded under the I/O Group.)

- Fix the state of the Flag, Control, and Overflow bits.
- Test the state of the Flag and Overflow bits (i.e., skip if set or clear, as specified).

- c. Enter data from a specific device into the A or B Registers.
- d. Output data to a specific device from the A or B Registers.
- e. Halt the program.

Input/Output instructions are recognized by the computer when the four most significant bits of the instruction word are 1000 and Bit 10 is a one. The codes and mnemonics for all 17 instructions are given in Figure 2.8 (the MAC instruction is not counted as a basic instruction). All Input/Output instructions are executed in one phase.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
	TYPE 3				A/B	*	H/C	INSTRUCTION				SELECT CODE				
MAC	1	0	0	0	0											
HLT	1	0	0	0	1			0	0	0						
STF	1	0	0	0	1	0		0	0	0	1					
CLF	1	0	0	0	1	1		0	0	0	1					
SFC	1	0	0	0	1	0		0	0	1	0					
SFS	1	0	0	0	1	0		0	0	1	1					
MIA	1	0	0	0	0	1		1	0	0						
MIB	1	0	0	0	1	1		1	0	0						
LIA	1	0	0	0	0	1		1	0	1						
LIB	1	0	0	0	1	1		1	0	1						
OTA	1	0	0	0	0	1		1	1	0						
OTB	1	0	0	0	1	1		1	1	0						
STC	1	0	0	0	0	1		1	1	1						
CLC	1	0	0	0	1	1		1	1	1						
STO	1	0	0	0		1	0	0	0	1	0	0	0	0	0	1
CLO	1	0	0	0		1	1	0	0	1	0	0	0	0	0	1
SOC	1	0	0	0		1		0	1	0	0	0	0	0	0	1
SOS	1	0	0	0		1		0	1	1	0	0	0	0	0	1

* Identifies Macroinstructions (0) or standard Input/Output instructions (1).

Figure 2.8. Input/Output Instructions

Note that Bit 11, where relevant, specifies A or B Register; otherwise it may be one or zero without affecting the instruction, although the Assembler will assign zeros (as shown). Bit 9, where not specified, offers the choice of Holding (0) or Clearing (1) the device Flag after execution of the instruction. (Exception: the H/C bit associated with the last two instructions in this list Holds or Clears the Overflow bit instead of the Flag bit.) Bits 8, 7, and 6 identify the instruction; some of the instructions, however, require additional specific bits for the complete code. Bits 5 through 0 form Select Codes to make the instruction apply to one of up to 64 input/output devices or functions.

The MAC instruction listed in Figure 2.8 is available to provide up to 2048 entries to macroinstruction subroutines. Since it is used only by special options and special software, MAC is not counted as one of the 70 basic machine instructions. The basic HP computer will treat MAC as a No-Operation (NOP) instruction.

HLT. Halt. Stops the computer, and Holds or Clears the flag (according to Bit 9) of any desired input/output device (Bits 5 through 0). The HLT instruction has the same effects as the HALT push-button: the HALT switch lights, all front-panel control switches are enabled, and no interrupts may occur. The HLT instruction will be displayed in the T-Register, and the P-Register (on 2115A/2116B Computers) will normally indicate the HALT location plus one.

STF. Set Flag. Sets the input/output Flag of the selected device, thus causing an interrupt during the next machine phase if the interrupt system is enabled, and the corresponding Control bit is set. The interrupt system itself is enabled by an STF instruction with a Select Code of 6 zeros (octal 00).

CLF. Clear Flag of selected device. Resets the Flag, thus permitting the device to present another Flag when ready again. A CLF with a Select Code of 6 zeros (octal 00) disables the entire interrupt system; this does not affect the status of individual input/output Flags.

SFC. Skip if Flag Clear. Causes the computer to skip the next instruction if the Flag bit of the selected device is zero (i.e., the device is not ready).

SFS. Skip if Flag Set. The next instruction is skipped if the Flag bit of the selected device is one (i.e., the device is ready).

MIA. Merge Input into A. The contents of the Input/Output Buffer associated with the selected device are merged ("inclusive or") into the A-Register.

MIb. Merge Input into B. The contents of the Input/Output Buffer associated with the selected device are merged ("inclusive or") into the B-Register.

LIA. Load Input into A. The contents of the Input/Output Buffer associated with the selected device are loaded into the A-Register. Previous contents of the A-Register are lost.

LIB. Load Input into B. The contents of the Input/Output Buffer associated with the selected device are loaded into the B-Register. Previous contents of the B-Register are lost.

OTA. Output from A. The contents of the A-Register are loaded into the Input/Output Buffer associated with the selected device. If the Buffer is less than 16 bits in length, the least significant bits of the A-Register normally are loaded. (Some exceptions exist, depending on the type of output device.) A-Register contents are not altered.

OTB. Output from B. The contents of the B-Register are loaded into the Input/Output Buffer associated with the selected device.

STC. Set Control bit of the selected device. This commands or prepares the device to perform its input or output function, and enables its Flag bit to interrupt the program being run (provided the program is not disabling the interrupt system).

CLC. Clear Control bit of the selected device. This prevents the device from interrupting. A CLC instruction with a Select Code of 00 (octal) clears all Control bits, effectively turning off all input/output devices. CLF 00 may be combined with this to additionally turn off the interrupt system.

STO. Set Overflow. The Overflow bit remains set until cleared by one of the following three instructions.

CLO. Clear Overflow. Resets the Overflow register.

SOS. Skip if Overflow Set. If the Overflow register is set, the next instruction of the program is skipped. Use of the H/C bit will Hold or Clear the Overflow bit following execution of this instruction (whether the skip is taken or not).

SOC. Skip if Overflow Clear. If the Overflow register is clear, the next instruction of the program is skipped. Use of the H/C bit will Hold or Clear the Overflow bit following execution of this instruction.

Table 2.3 consolidates all of the instructions contained in Figures 2.5 through 2.8 for a quick, easy reference to all computer instructions. To use the table, simply find the mnemonic for the instruction and then read all coding bits on the same line as the mnemonic. Those bit locations specifying A/B, Z/C, etc. must be programmed to contain a 0 or 1, respectively, as applicable to the instruction to be performed. Also, all mnemonics for instructions involving the A or B Register contain an asterisk in place of the appropriate A or B letter. Each of these instructions use a 0 or 1 in bit position 11 (depending on whether the A or B Register is used) as well as those bits specified on the same line as the mnemonic. The instructions in the Shift-Rotate Group (except CLE and SL*) require a "one" in bit positions 4 and/or 9 to be executed. A "one" in bit position 4 enables an instruction defined by bits 0, 1, and 2; bit position 9 enables an instruction defined by bits 6, 7, and 8. If both bit positions 4 and 9 contain a "one", the instruction defined by bits 6, 7, and 8 is executed first.

2.8 DATA FORMATS

Data is represented in two's complement form internally in the HP computer. The basic format for arithmetic operations on numerical data is defined in Figure 2.9. The data is assumed to be an integer (binary point to the right of Bit 0), and is positive if the sign bit is zero, or negative if one. The largest possible positive number (in octal) is +77777, or (in decimal) +32767; the largest possible negative number is -100000 (octal) or -32768 (decimal). Other possible formats, including packed data words, double-length fixed point, and floating point representations, are defined in standard software packages.

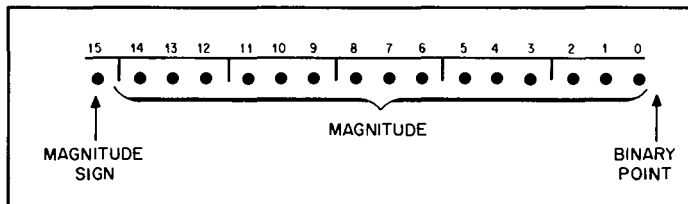


Figure 2.9. Basic Data Format

2.9 INPUT/OUTPUT SPECIFICATIONS

2.9.1 INPUT/OUTPUT SYSTEM DESIGN

GENERAL. Information is transferred into the computer from an external device, or out of the computer to an external device, by

way of its input/output capability, termed the input/output system. A transfer of information is initiated by a signal from a device indicating that it is ready for input or output. The transfer occurs by the process of interrupting a running program (which could be either a problem-solving program, or a program specifically designed to transfer data). The interrupt directs the computer to a location in memory uniquely associated with the interrupting device. This location in turn directs the computer to a program routine (a "service routine", which must previously have been stored in memory), and this routine will contain instructions which effect the actual transfer of information. Since interrupts can occur at almost any time, including during the service routine of an earlier interrupt, a priority network is present in the computer to establish the sequence in which interrupts are serviced. As shown in Figure 2.10, the input/output system capability is physically divided so that part of the capability (including the priority network and the identical hardwiring for optional plug-in card slots) is an integral part of the HP computer main unit. The remaining part is provided by Input/Output Options, which will include the plug-in interface cards and cables for specific devices, and the appropriate software drivers and diagnostic programs. The interface cards may be plugged into any of the identical input/output slots, depending on the desired priority rating. Each combination of interface card and device, when plugged into the computer, constitutes an input/output channel.

NUMBER OF CHANNELS. The coding structure of input/output instructions (Figure 2.8) allows 6 bits for a Select Code, making it possible to specify a total of 64 channels and functions (2^6). Of this total, four Select Codes are assigned to non-interrupting functions (Interrupt System Enable/Disable, Switch Register/Overflow, and initialization of two Direct Memory Access channels). Of the 60 remaining channels and functions with interrupt capability, four interrupt assignments are reserved for internal processor functions: Power Fail, Memory Protect (2116B only)/Memory Parity, and the interrupt assignments of the two Direct Memory Access channels (2115A/2116B only). Thus, 56 possible channels remain for input/output devices. Table 2.4 indicates the number of these remaining 56 channels accommodated by the basic unit of each computer and those accommodated by the extender modules used with each computer.

Table 2.3. Consolidated Coding Table

[illegible]

0	ASG	000	A/B 1 A/B 1 A/B 1	CL* 01 CM* 10 CC* 11	CLE 01 CME 10 CCE 11	SEZ SS* SL*	IN* SZ* RSS								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

MAC AND INPUT/OUTPUT INSTRUCTIONS

1	MAC	000	A/B 0												
1	IOG	000	A/B 1	H/C	HLT	000									
				0	STF	001									
				1	CLF	001									
				1	SFC	010									
				0	SFS	011									
			A/B 1	H/C	MI*	100									
			A/B 1	H/C	LI*	101									
			A/B 1	H/C	OT*	110									
			0	1	STC	111									
			1	1	CLC	111									
				0	STO	001							000		001
				1	CLO	001							000		001
				1	SOC	010							000		001
				1	SOS	011							000		001

← Select Code →

- Notes: 1) * = A or B. Use with bit 11 as 0 (A-Register) or 1 (B-Register).
 2) D/I, A/B, Z/C, D/E, H/C coded: 0/1.
 3) †CLE: Only this bit is required.
 4) †SL*: Only this bit and bit 11 (A/B as applicable) are required.

Table 2.4. Computer I/O Channels

Computer	Basic Unit	2150A/B Extender Module	Total
2116B	16	32	48
2115A	8	32	40
2114A	8	--	*8

*56 channels are possible using Multiplexed I/O option.

INTERFACE COMPONENTS. Each plug-in interface card normally includes the following components, shown in Figure 2.11.

a. An Input/Output Buffer consisting of up to 16 flip-flops for temporary storage of data to be transferred in or out, so that it is not necessary to tie up a working register during the relatively long transfer periods. The actual number of Buffer bits, from 1 to 16, will depend on the device for which the interface is intended. Data is transferred to the Buffer from the A or B Registers by OTA or OTB instructions, and is brought in to the A or B Registers from the Buffer by LIA, LIB, MIA, or MIB instructions. If the Buffer is less than 16 bits in length, data is transferred to or from the least significant bits of the A or B Registers.

b. An input/output Flag flip-flop, which will be set by a signal from the external device when the device has completed an operation. The Flag may also be set, if desired, by program instruction (STF). Once set, the Flag remains set until reset by a clear instruction (CLF or H/C bit). Provided it is itself not inhibited by the set Flag of a higher priority device, or otherwise disabled (see step "c"), the Flag, when set, inhibits all interrupts for devices having lower priority. It will cause an interrupt after the current machine phase. Successive interrupts for one device may occur on receipt of a number of Flag signals without executing a Clear Flag instruction, thus making it possible to inhibit lower priority devices indefinitely until a desired number of high-priority transfers have been completed. The Flag can be set and cleared even if its interrupt capability is inhibited or disabled, and may be tested by SFS or SFC instructions.

c. A Control flip-flop to command or enable the external device to perform its input or output operation. In addition, the Control bit controls the interrupt capability for that particular device; i.e., unless the Control flip-flop is set, a received Flag cannot cause an interrupt, nor can it inhibit the interrupt capability of any other device in the priority string. Thus the Control bit, when set, effectively "turns on" the individual input/output channel.

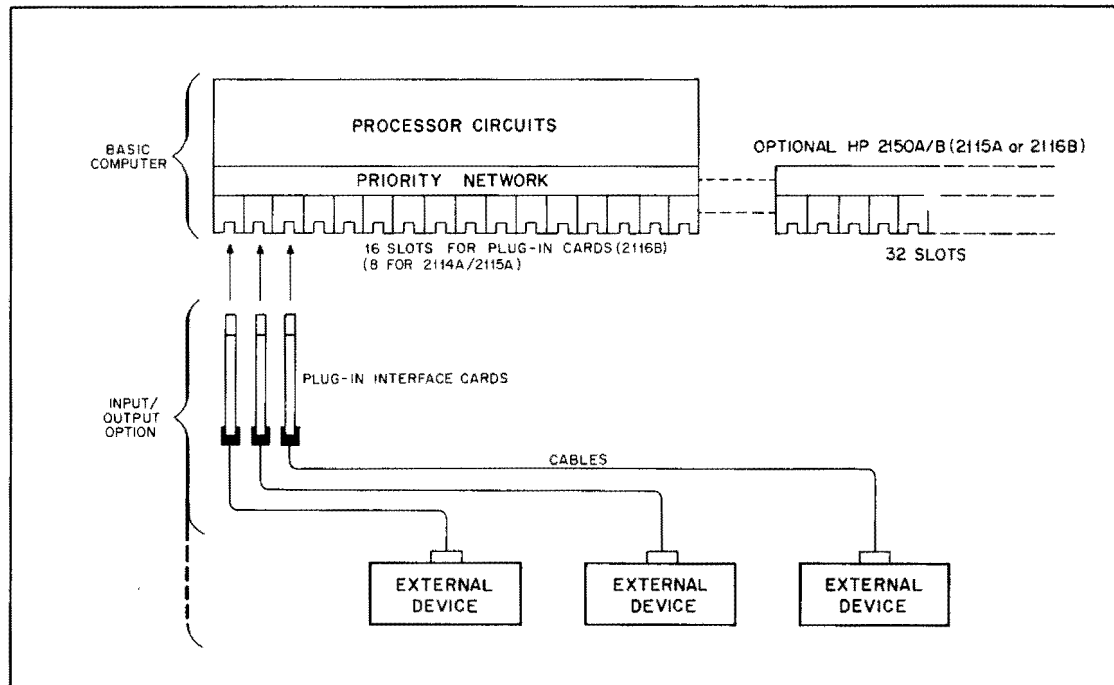


Figure 2.10. Input/Output Design Arrangement

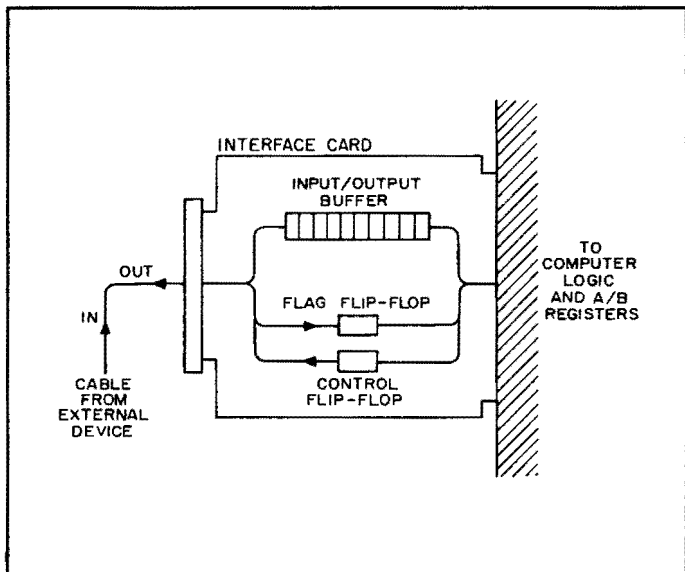


Figure 2.11. Components of Typical Input/Output Interface Card

PARTY LINE INPUT/OUTPUT. This option permits the interfacing of customer designed and standard devices to the 2116B, 2115A or 2114A Computer using only those input/output lines normally assigned to a single peripheral device. The input/output lines from the computer are bussed to all devices on the party line so the devices appear as a single device to the computer. (See Figure 2.12.) Since all lines are identically available to all devices, each device must have its own controller (furnished by the customer). The controller must decode device address and command information, send status information to the computer, and maintain overall control of the device to which it is connected. Two Microcircuit Interface cards are required to implement party line I/O. The cards are identical and each card contains two 16-bit "or-tied" registers as well as standard control and interrupt circuits common to all HP input/output interface cards. The receivers and drivers on each card are microcircuit, rather than discrete, components for ease of interfacing. The cards provide an Encode (take action) signal to the party line devices, the facilities to recognize a Flag (action completed) signal from a party line device, and 16 lines for input/output data. The party line interface cards plug into any two adjacent computer I/O slots; the remaining slots can be used to interface

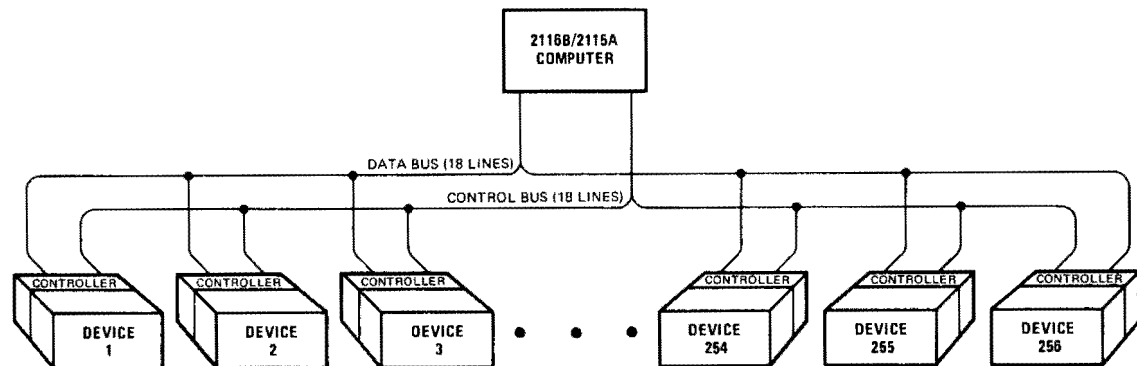


Figure 2.12. Party Line Block Diagram

standard peripheral devices. The slot positions establish the priority of the party line in relation to the other devices connected to the computer. Data transfer rates of 40 kHz are possible under a non-interrupt mode, while rates are limited to 10 to 12 kHz under interrupt control.

2114A MULTIPLEXED INPUT/OUTPUT. This option permits the connection of up to 56 devices to the 2114A Computer. A special interface card, the Multiplexer Data card, plugs into a computer I/O slot and operates in conjunction with an optional I/O Control card (which replaces the standard I/O Control card) to provide 56 signal lines (see Figure 2.13). These signal lines handle addressing, provide the control signals, receive interrupt signals, and input/output up to 16 bits of data. All decoding of device addresses and I/O instructions is performed by the computer. The I/O device requests service of the computer by sending its binary coded address to the computer on six interrupt request lines. Interrupt requests are relayed to specific memory locations through the existing computer interrupt system for processing. Interfaces between the external devices and the computer are customer furnished but interfacing is simplified since all necessary signals are supplied by the computer.

SELECT CODE ASSIGNMENTS. As mentioned previously, Bits 5 through 0 of the Input/Output instructions form a Select Code to specify one of 64 possible input/output devices or functions. Of the 64 Select Codes, some are reserved for specific uses while others are available for assignment to any optional input/output device. Table 2.5 lists these assignments, and gives the corresponding interrupt location (i.e., the location containing the instruction to be executed when interrupt occurs). The first five (octal Codes 00 through 04) are reserved for non-interrupting functions.

2.9.2 INTERRUPT STRUCTURE

OPERATION. On computer command (Set Control instruction STC), one or more external devices begin their read or record operations, putting data into (input) or taking data from (output) the Input/Output Buffer on each individual interface card. During this time, the computer may continue running a program, or may be programmed into a waiting loop to wait for a specific device. On completion of the read or record operations, each device returns an "operation completed" signal (Flag) to the computer. The Flags are passed through a priority network which allows only one device to be serviced regardless of the number of Flags simultaneously present. The Flag with the highest priority causes an interrupt at the end of the current machine phase, switching the computer into the Interrupt phase, except under any of the following circumstances.

- a. Interrupt System disabled or device interrupt disabled.
- b. Computer in HALT mode. SINGLE CYCLE pushbutton cannot step the computer into the Interrupt phase.
- c. JMP Indirect or JSB Indirect not fully executed. These instructions inhibit all interrupts until the instruction (plus one phase of the succeeding instruction) is completed.
- d. Instruction in an interrupt location not fully executed, even if of lower priority. Any interrupt inhibits the entire interrupt system until one instruction (plus one phase of the succeeding instruction) has been completed. (In the case of a multi-level indirect instruction, the interrupt system will be re-enabled after two Indirect phases; JMP Indirect and JSB Indirect are exceptions and will be fully executed.)
- e. Direct Memory Access Option in process of transferring data (2116B/2115A). Exception: power failure control can interrupt a DMA transfer.
- f. The current instruction is one which may affect the priorities of input/output devices (STC, CLC, STF, CLF). The interrupt in this case must wait until the end of the succeeding machine phase.

When interrupt occurs, the computer puts the Select Code number of the interrupting device into the M-Register (with extra zeros to specify page Zero), thus causing the next instruction to be read from the memory location having the same number as the Select Code. This location in memory is referred to as the "interrupt location", and is reserved for that particular device. Example: a device specified by a Select Code of 10 will interrupt to (i.e., cause execution of the contents of) memory location 00010. The instruction in the interrupt location will usually be a jump (JSB) to an input or output subroutine.

To prevent external devices from running when computer power is first turned on, turn-on of the POWER switch automatically clears all Control bits, resets the Interrupt System Enable flip-flop (disabling the interrupt system), and sets all device Flags. Pressing the PRESET pushbutton accomplishes the same function when the computer is on (but not when running, since the control switches are disabled). Therefore, before any device can operate with the computer, it is necessary for the program to set the Interrupt System Enable flip-flop and, depending on the type of device, clear the individual Flag bit and/or set the individual Control bit.

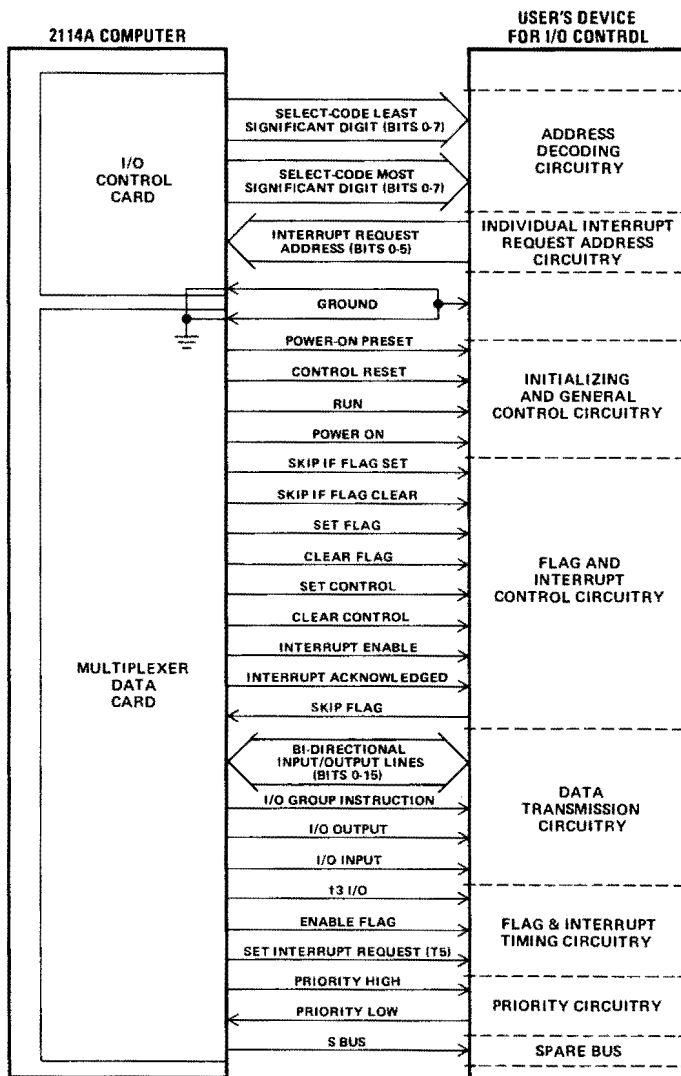


Figure 2.13. Multiplexed I/O Signal Lines

Table 2.5. Select Code Assignments

SELECT CODE (OCTAL)	INTERRUPT LOCATION	ASSIGNMENT
00	None	STF \emptyset turns Interrupt System on CLF \emptyset turns Interrupt System off
01	None	Switch Register: LIA/B, MIA/B OTA/B (2114A only) Overflow Register: STO, CLO, SOC, SOS
02	None	Initialize DMA Channel 1 (Select Code 6): 2116B, 2115A
03	None	Initialize DMA Channel 2 (Select Code 7): 2116B, 2115A
04	00004	Power Fail Interrupt Interrupt Source Identification (LIA/B 4 = Select Code of last interrupting device)
05	00005	Memory Protect Interrupt (2116B only) Parity Error Interrupt (LIA/B 5 = address of offending location. If bit 15 = "1", parity error interrupt. If bit 15 = "0", memory protect interrupt.)
06	00006	DMA Channel 1 Completion Interrupt: 2116B, 2115A
07	00007	DMA Channel 2 Completion Interrupt: 2116B, 2115A
10 thru 77	00010 thru 00077	I/O Device, highest priority thru I/O Device, lowest priority

INPUT INTERRUPT. The typical operation sequence for an input interrupt involves the following steps.

a. A STC instruction, usually accompanied by CLF, sends a command (equivalent to "read", "encode", or "reset" in HP digital measuring equipment) to the external device.

b. The device reads its input, then puts the data into the Input/Output Buffer on the interface card.

c. Simultaneously, the device supplies a Flag signal (equivalent to "record" or "print" commands in HP digital measuring equipment) to the computer.

d. The Flag is converted to an interrupt request by the device interface card.

e. The resulting interrupt causes a service subroutine for that device to begin, temporarily suspending operation of the main program.

f. The service subroutine enters data from the Buffer into the A or B Register, then returns control to the main program.

OUTPUT INTERRUPT. The typical operation sequence for an output interrupt involves the following steps.

a. An OTA or OTB instruction puts data from the A or B Register into the Input/Output Buffer on the interface card.

b. A STC instruction sends a command (equivalent to "record" or "print" commands in HP digital recording equipment) to the external device.

c. The device accepts (records) the data currently in the Buffer.

d. After the data has been accepted, the device returns a Flag signal (equivalent to the end of a "holdoff" or "inhibit" signal in HP digital recording equipment) to the computer.

e. The Flag is converted to an interrupt request by the device interface card.

f. The interrupt causes a service subroutine for that device to begin.

g. The service subroutine loads new data into the Buffer, repeating the sequence.

PRIORITY. The priority network gives highest interrupt priority to Select Code 04, reserved for power failure control interrupt, and decreasing priority to Select Codes in order from 05 through 77. The transfer of data by the two optional Direct Memory Access (DMA) channels (which transfer data directly to and from memory by inserting a special memory cycle, rather than by interrupt to a service subroutine) effectively have priorities between Select Codes 06 and 07, since they can inhibit all interrupts except power failure

control. DMA Channel 1 has priority over DMA Channel 2. (DMA is not used in 2114A.)

A set Flag inhibits all Flags below it on the priority string, and once this Flag is cleared, the next lower can then interrupt. A service subroutine for any device can be interrupted by a higher priority device; then, after the higher Flag is cleared, the subroutine may continue. In this way, it is possible for several service subroutines to be in a state of interruption at one time; each will be permitted to continue when the higher priority Flag is cleared. All service subroutines normally end with a Jump-Indirect (JMP, I) instruction to return the computer to the point of interrupt.

TRANSFER RATE. Up to 60,000 transfers per second, limited by length of service subroutine, are possible with the 2116B. Up to 47,000 transfers per second are possible with the 2114A and 2115A.

2.10 PROCESSOR OPTIONS

The following Options are all capable of being installed in the field. They consist of one or more plug-in cards, and in the case of Option M4, an additional memory module as well. Other processor Options are available, as either standard or custom modifications; consult the applicable HP Computer Technical Data sheet or a Hewlett-Packard field office.

8K MEMORY. Option M4 (2114A and 2115A only). Comprises a set of memory addressing cards and an 8192-word memory module. The module replaces the 4096-word memory module in the computer mainframe.

16K MEMORY. Option M5 (2116B only). Adds 8192-word memory to the 8192-word memory in the basic computer.

MEMORY PARITY CHECK. Option M2. Permits parity checking within memory. Odd parity is used. Option M2 consists of one plug-in card. A parity error may either cause the computer to halt or interrupt to address 05.

EXTENDED ARITHMETIC. Option M9 for 2115A and 2116B computers. Permits Integer Multiply, Integer Divide, Double Load, Double Store, Long Shifts, and Long Rotations to be performed by hardware instead of program subroutines.

2.11 INPUT/OUTPUT OPTIONS

Input/output options for HP computers, identified by Interface Kit accessory numbers, consist of a combination of plug-in cards, in-

terconnecting cables, and appropriate software. Table 2.6 summarizes some of the available standard input/output options, and the following paragraphs briefly describe the capabilities added by these options.

In many cases, as indicated in Table 2.6, an optional Interface Kit is designed to operate with more than one kind of peripheral device, or with different versions of a device. An example is the HP 12544A Interface Kit, for which the HP 5245L Electronic Counter and HP 2801A Quartz Thermometer are listed as typical devices; however, the range of compatible devices can include other 8-digit devices of similar output.

In some cases, one peripheral may be associated with more than one interface. An example is the HP 2401C Integrating Digital Voltmeter, which requires one interface (HP 12541A) to transfer its data into the computer, and another interface (HP 12533A) to accept function commands from the computer.

Most Input/Output Options require only one card. This card by itself has no definite Select Code assignment or interrupt priority. Plugging the card into any of the general purpose input/output slots of the computer, each of which has a Select Code assignment, automatically gives the external device an interrupt priority, according to the Select Code of the slot.

As shown in Figure 2.14, each of the input/output slots in the computer actually has two Select Codes available, although usually only one is used by the interface cards. If a slot between interface cards is unused, a Priority Jumper card must be inserted in the empty slot to maintain interrupt priority continuity. There can be no gaps in the priority string; continuity is required from Select Code position 10 up to the last used Select Code.

To use more interface cards than the computer can accommodate in its Input/Output slots requires an extender module. The HP 2150A Extender Module, with Option M1, can be used with the 2115A or the 2116B Computer and allows the user to install up to 32 additional interface cards. (Eight I/O slots in the 2115A plus 32 in the 2150A gives a total of 40 I/O slots; 16 slots in the 2116B plus 32 in the 2150A gives a total of 48.) Option M1 for the 2150A comprises four plug-in buffer cards and two interconnecting cables.

TELEPRINTER INPUT/OUTPUT. The simplest configuration of an HP computer system is provided by a combination of a computer and an HP 2752A Teleprinter (modified Teletype ASR-33) and accessory Interface Kit 12531B. The Teleprinter combines a typewriter, punched tape reader and tape punch. Data and instructions

may be entered from punched tape or the keyboard. Output information is recorded on the typewriter, and may be recorded simultaneously on punched tape. The Teleprinter operates at 10 characters/second for both data entry and data recording. Where heavy use of the Teleprinter is anticipated, exceeding 5 hours per day or 30 hours per week, a heavy duty HP 2754A Teleprinter (modified Teletype ASR-35) is recommended. This device uses the same interface. The HP 2752A and HP 2754A Teleprinters perform the same functions and operate at the same speed.

HIGH-SPEED PUNCHED TAPE INPUT. For rapid entry of punched tape programs and data into an HP computer, a 300 characters per second HP 2737A Punched Tape Reader, with its Interface Kit 12532A, is available. The reader is equipped with a container for the tape to be read.

HIGH-SPEED PUNCHED TAPE OUTPUT. Computer data output can be recorded (asynchronously) on punched tape at 120 characters per second with an HP 2753A Tape Punch and Interface Kit 12536A. This device includes a tape spooler, which accepts approximately 1000 feet of tape.

7-CHANNEL MAGNETIC TAPE INPUT/OUTPUT. Interface Kit 12538A enables the computer to record on and read from 1/2 inch, 7-channel, NRZI, IBM-compatible magnetic tape with HP H26-2020A and HP H26-2020B Magnetic Tape Units. The HP H26-2020A Magnetic Tape Unit reads and records at 200 bpi density. Tape speed is 30 ips, providing a data transfer rate of 6000 characters/second. With the dual-density model HP H26-2020B, the Tape Unit operates at either 200 or 556 bpi density, switch-selectable. Tape speed is also 30 ips, providing a data transfer rate of 16,700 characters per second when set to operate at 556 bpi.

9-CHANNEL MAGNETIC TAPE INPUT/OUTPUT. Interface Kit 12559A enables the 2115A or 2116B Computer with DMA Option M11 to record on and read from 1/2-inch, 9-channel, NRZI, IBM-compatible magnetic tape with the HP H01-3030G Magnetic Tape Unit. This unit reads and records at 800 bpi density. Tape speed is 75 ips, providing a data-transfer rate of 60,000 characters per second.

DISC MEMORY. The 2757A Disc Memory and Interface Kit 12561A provide a high capacity, random access memory extension for the 2115A and 2116B Computers with DMA Option M11. The disc memory has a storage density of 174,000 or (optional) 348,000 16-bit words, with an average access time of 16.7 milliseconds. Data can be stored and retrieved at a rate of 176,000 16-bit words per second.

Table 2.6. Input/Output Options

I/O OPTION	CAPABILITY	INTERFACE KIT	PERIPHERAL	OPTION NO.
TELEPRINTER INPUT/OUTPUT	System records on typewriter and punched tape, and inputs from keyboard and punched tape, at 10 characters/sec. (ASCII code)	12531B	HP 2752A Teleprinter (modified Teletype ASR-33)	K1
HEAVY DUTY TELEPRINTER INPUT/OUTPUT	Similar to Option K1 above except heavy-duty Teleprinter is furnished. Recommended where use exceeds 5 hrs/day or 30 hrs/week.	12531B	HP 2754A Teleprinter (modified Teletype ASR-35)	K2
HIGH-SPEED PUNCHED TAPE INPUT	System inputs from punched tape at 300 characters/sec. (ASCII code)	12532A	HP 2737A Punched Tape Reader (with accessory 12525A Tape Holder)	K3
HIGH-SPEED PUNCHED TAPE OUTPUT	System records on punched tape at 120 characters/sec. (ASCII code)	12536A	HP 2753A Tape Punch	K5
INCREMENTAL MAGNETIC TAPE OUTPUT (1200 FT REELS)	System records on IBM-compatible, 1/2 inch, 7 channel NRZI tape. Bit density 200 bpi. Recording speed 400 characters/sec. Reel capacity 1200 feet.	12537A	Kennedy 1406 Incremental Tape Transport	K6
INCREMENTAL MAGNETIC TAPE OUTPUT (2400 FT REELS)	Similar to Option K6 above, but reel capacity 2400 feet.	12537A	Kennedy 1506 Incremental Tape Transport	K7
LOW DENSITY MAGNETIC TAPE INPUT/OUTPUT	System records on, and reads from, IBM-compatible 1/2 inch 7-channel NRZI tape. Bit density 200 bpi. Speed 30 ips.	12538A*	HP (H26) 2020A Magnetic Tape Unit	K8
DUAL DENSITY MAGNETIC TAPE INPUT/OUTPUT	Similar to Option K8 above, but System records and reads at both 200 and 556 bpi.	12538A*	HP (H26) 2020B Magnetic Tape Unit	K9
DISC MEMORY (3.2 MEGABITS)	Provides storage for 174,080 words in 64-word addressable sectors. Transfer rate 3 megabits/second. Access time 33 ms max. Requires DMA.	12561A*	HP 2757A Disc Memory with HP 2756A Disc Memory Power Supply.	K33
DISC MEMORY (6.4 MEGABITS)	Similar to Option K33 above, but 348,160 words	12561A*	HP 2757A-M1 Disc Memory with HP 2756A Disc Memory Power Supply.	K34
9-CHANNEL MAGNETIC TAPE INPUT/OUTPUT	System records on, and reads from, IBM-compatible 1/2 inch 9-channel tape. Bit density 800 bpi. Speed 75 ips. Requires DMA.	12559A*	HP (H01) 3030G Magnetic Tape Unit	K35
AUXILIARY POWER	Supplements internal I/O power supply.	—	HP 2160A Power Supply	K36
		*Use 2 I/O slots		

- Following I/O Options K10 through K42 do not include the peripheral.

TIME BASE GENERATOR	Generates real time intervals in decade steps from 100 μ s to 1000 sec (derived from crystal oscillator). Used as source of timed interrupts for software clock.	12539A	None required	K10
DATA PHONE INTERFACE	Interfaces System with Bell System Data Phone service.	12540A	Bell System Data Set 103A	K11
DIGITAL VOLTMETER DATA INPUT (HP 2401C)	System accepts bcd (8421) data output from HP 2401C Integrating Digital Voltmeter.	12541A	HP 2401C Integrating Digital Voltmeter (with mod. M21)	K12
DIGITAL VOLTMETER DATA INPUT (HP 3440A)	System accepts bcd (8421) data output from HP 3440A Digital Voltmeter.	12543A	HP 3440A Digital Voltmeter (with 8421 output)	K14
COUNTER/THERMOMETER DATA INPUT (8 DIGITS)	System accepts bcd (8421) data output from 8-digit electronic counter and the quartz thermometer.	12544A	HP 5245L Electronic Counter (with option 02). HP 2801A Quartz Thermometer (with mod. M6)	K15
DIGITAL VOLTMETER PROGRAM OUTPUT (HP 2401C)	Enables System to select function, range, etc., of HP 2401C Integrating Digital Voltmeter.	12533A	HP 2401C Integrating Digital Voltmeter (with mods. M21, 146)	K20
DIGITAL VOLTMETER PROGRAM OUTPUT (HP 2401C/2411A)	Same as Option K20 above, except HP 2411A Data Amplifier included.	12550A	HP 2401C (M21, 146) Digital Voltmeter and HP 2411A Data Amplifier	K21
CROSSBAR SCANNER PROGRAM OUTPUT (HP 2911)	Enables System to select channel, settling delay, for HP 2911 Guarded Crossbar Scanner.	12535A	HP 2911A Guarded Crossbar Switch and 2911B (M33) Scanner Control.	K23
GENERAL PURPOSE DUPLEX REGISTER	Dual 16-bit flip-flop register. Permits bidirectional transfer of information between System and external devices. (Accessory kit includes 48-pin mating connector.)	12554A	Determined by user.	K24
RELAY OUTPUT REGISTER	Provides 16 form A contacts for operating external devices. (Accessory kit includes 48-pin mating connector.)	12551B	Determined by user.	K25
D-A CONVERTER	Provides two D-A conversion channels, 8 bits/channel.	12555A	Determined by user.	K42
MULTIPLEXED I/O	Permits up to 56 devices to be multiplexed into one I/O channel.	12595A	Determined by user.	K43

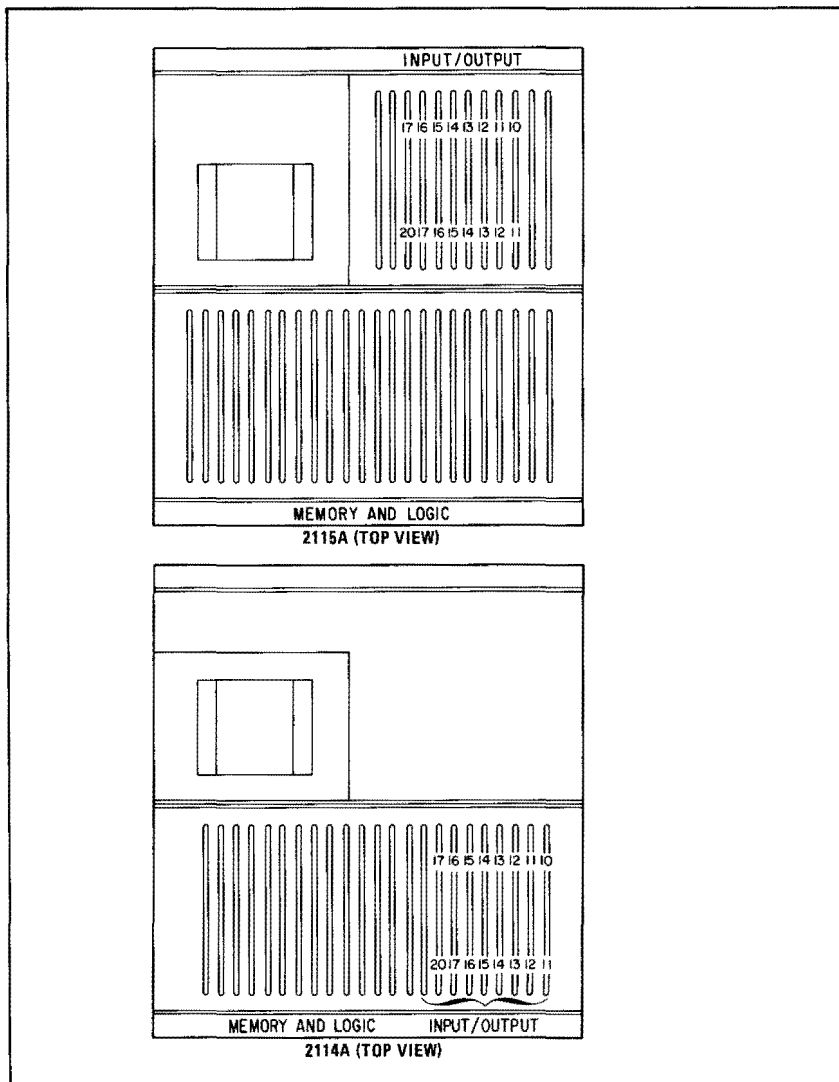
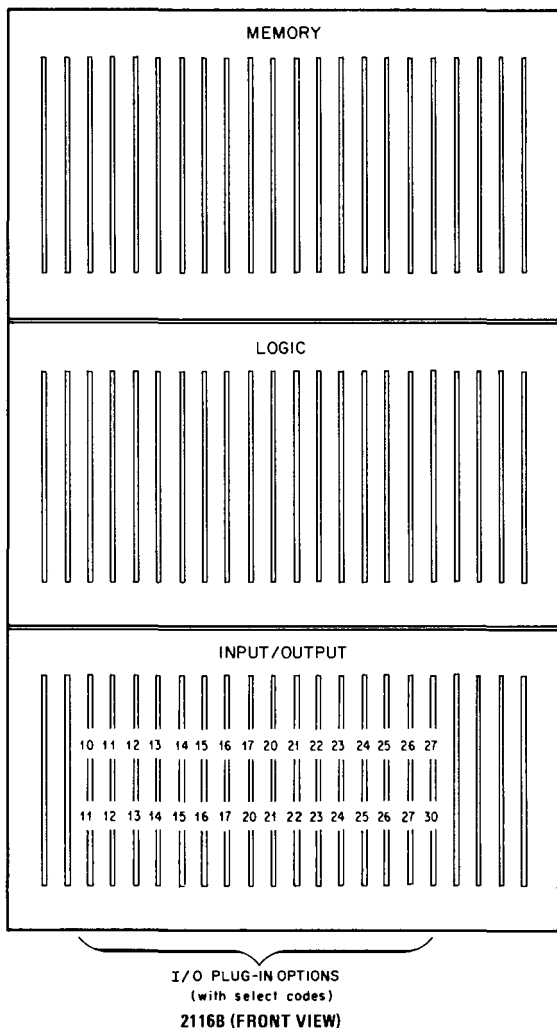


Figure 2.14.



Input/Output Option Locations

DATA SOURCE INTERFACES. A data source interface card, with special cables, is available to permit HP computers to operate with virtually all Hewlett-Packard instruments providing a digital data output in binary or binary-coded decimal, positive or negative-true form. This encompasses a very broad variety of instruments, principal examples being digital voltmeters, electronic counters, nuclear scalars, and quartz thermometers. The same interface card, which accepts 32 bits (8 BCD digits), is used with all these instruments (one card for each instrument) but different interconnecting cables are involved. Therefore, for simplicity in assembling a system, the interface card coupled with the appropriate cable is listed as an option for a specific group of data sources, as Interface Kits 12541A through 12548A (see Table 2.6). Since these instruments require no modification to interface their data output with the HP computer, they can be ordered directly from the Hewlett-Packard catalog.

DIGITAL VOLTMETER PROGRAMMERS. In measurement systems, when using digital voltmeters as data inputs to the HP computer, the computer may select voltmeter functions such as mode (dc/ac volts, ohms), range, and resolution (sample period). Interface Kit 12533A comprises the interface card for this purpose, together with the interconnecting cable for the HP 2401C Integrating Digital Voltmeter. Interface Kit 12534A furnishes the same card, but a different cable, for the HP 3460A Digital Voltmeter. Interface Kit 12567A permits function programming of the 2402A Integrating Digital Voltmeter when it is used in conjunction with the 2911 Guarded Crossbar Scanner. No additional interface circuitry is required when these voltmeters are used with their accessory signal amplifiers and signal converters. The Digital Voltmeter Programmer interface card provides 20 output lines, each capable of switching 200 ma from an external 35-volt negative supply.

CROSSBAR SCANNER PROGRAMMER. For multiple-channel analog measurements with the HP computer, a scanner is necessary to inter-connect the signal input channels with one or more analog-to-digital converters, as required. Any one signal path is enabled at a time on command from the computer, which selects the input channel to be measured and diverts it to the appropriate A-D converter. It also initiates a "measurement delay" before sampling (encoding) commences, if such is necessary for converter settling. The interface card and cable for programming an HP 2911 Guarded Crossbar Scanner are provided under Interface Kit 12535A.

TIME BASE GENERATOR. The Time Base Generator, Interface Kit 12539A, provides the computer with a train of program interrupts at real time intervals. It consists of a crystal oscillator and decade frequency dividers, contained on one I/O card. The inter-

val between interrupts is computer-selectable in decade steps from 100 microseconds to 1000 seconds (approximately 16 minutes). Time-of-day, if required, is obtained from this real time reference by software. Accuracy is better than 1/2 second per 24-hour day, under typical operating conditions, (aging rate $< 2/10^6$ per week; temperature effect $< 2/10^5$, $+15^\circ$ to 35°C).

INCREMENTAL MAGNETIC TAPE OUTPUT. Computer data output can be recorded on 1/2-inch magnetic tape in 7-channel, NRZI, IBM-compatible format with a Kennedy 1406 Incremental Magnetic Tape Transport when used with the HP 12537A Interface Kit. No provision is made for tape reading. Data is recorded with a density of 200 bpi, at a speed of 400 characters/second. The Kennedy 1406 uses side-by-side 8-1/2 inch reels to economize in panel height (12-1/4 inches). Reel capacity is 1200 feet of tape. Kennedy Model 1506, which uses the same Interface Kit, is essentially the same transport, but features 10-1/2 inch, 2400-foot capacity reels. (Panel height 24-1/2 inches.)

DATA-PHONE INTERFACE. Information can be transferred in or out of the computer over the telephone system with Interface Kit 12540A, which consists of the data-phone interface card and interconnecting cable to operate with a Bell Telephone Data Set 103A. Data transfer occurs at a rate of 10 eight-bit characters/second.

Any of the above Input/Output Options can be added or deleted, and service priorities changed, on a plug-in basis. No wiring changes to the computer are involved. Input/output software also is modular, and a software configurator is furnished which allows the user to change his software operating system to handle different hardware configurations with minimal programming effort.

2.12 SOFTWARE

2.12.1 GENERAL

Hewlett-Packard computers are supported by a full range of software, normally furnished in the form of punched paper tape. As standard accessories, the following software packages are supplied with all HP computers, unless additions or deletions are otherwise specified. All are operable with the minimum computer system configuration; i.e., 4K memory and Teleprinter input/output. Printed listings of these programs are furnished as part of the system documentation.

Basic Control System modules

Symbolic Editor

Assembler

FORTRAN Compiler
Program Library
System Input/Output Drivers
Hardware Diagnostics
ALGOL Compiler
BASIC Interpreter

Each of the software packages listed above consists in most cases of a number of individual tapes. The number of tapes furnished depends on the Options purchased with a system; Driver tapes and Test tapes are furnished as accessories to interface Options when purchased, either with the initial order or with field installation.

A listing of standard software for HP computers is published in the HP Software Catalog, available from Hewlett-Packard Sales Offices.

In addition to these standard tapes, two configured tapes, incorporating actual system device assignments, are furnished with the initial shipment, one for the System Input/Output Drivers and one for the Basic Control System. The System Input/Output (SIO) Drivers primarily provide input/output capability for the Assembler, Symbolic Editor, and FORTRAN compiler, but may also be used as desired in user's programs. The Basic Control System, on the other hand, is primarily intended to provide a complete software input/output system for user's programs. These two tapes are unique to each system, and do not have HP Accessory Numbers and are not listed in the Software Catalog. Subsequent reconfiguring of System Input/Output and the Basic Control System, if desired, is easily accomplished by the user, with the aid of supplied software (System Input/Output Dump, and Prepare Control System).

TAPE IDENTIFICATION. Each software tape is separately identifiable by description and HP Accessory Number, labeled on both the tape container and the tape itself. The letter at the end of the number identifies a particular version of the tape (e.g., B supersedes A). A detailed list of the software packed with the system is given in the Software Installation Record, supplied with the system documentation at the time of computer delivery. When ordering new or duplicate tapes (or documentation), the latest applicable version will automatically be furnished. Software is ordered through Hewlett-Packard Field Sales Offices.

2.12.2 HARDWARE DIAGNOSTICS

To assist the user in hardware troubleshooting, a Hardware Diagnostics package is furnished with all HP computers. The programs

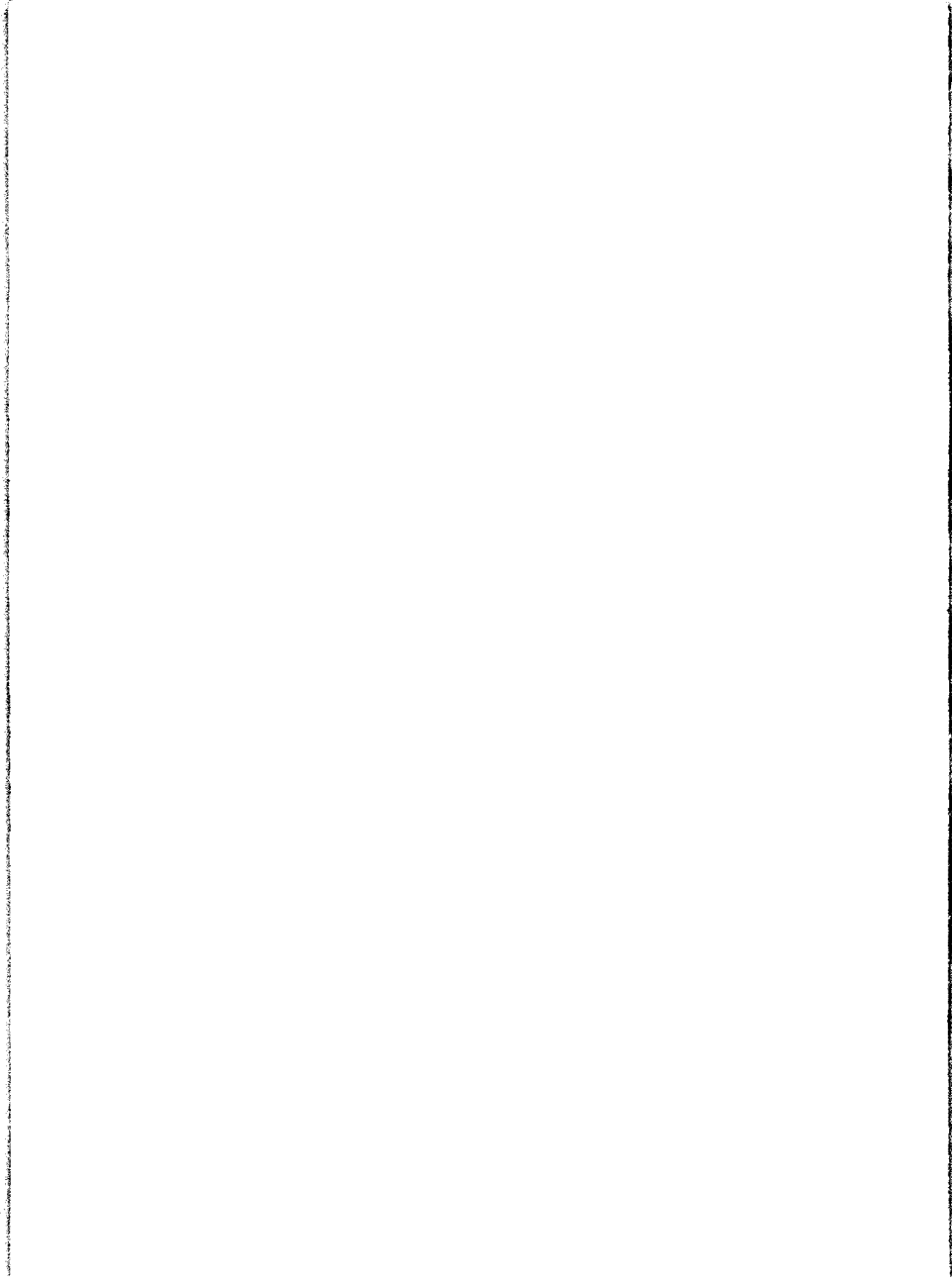
in this package are separate and independent, and are used in addition to the maintenance information given in the HP Installation and Maintenance Manual (Volume Two) provided with the computer. Each package consists of the diagnostic tapes, the listings for the tapes, and use instructions. The maintenance documentation gives procedures to determine that the hardware system is capable of accepting and using the Hardware Diagnostic programs. Then the supplied software may be loaded and run according to set procedures. Programs supplied are:

a. **Instruction Tests.** These tests check out all instruction codes in groups, halting the computer when an instruction fails to perform its function. The first test program checks out a few basic instructions (Alter-Skip), so that those instructions can be used by the next test program (Memory Reference), which in turn enables checking out the final group (Shift-Rotate).

b. **Memory Address Tests.** These tests verify the accessibility of all memory addresses for which the computer is configured (e.g., 4K, 8K, etc.). A Low-Core Test and a High-Core Test are supplied as separate test programs, so that the program may be loaded at the end of memory to check all core locations below the test block, or it may be loaded at the bottom of memory to check all higher locations. Each Test checks the addressing logic of a selectable section of memory, and halts when an error is detected. The display on the computer front panel is used to identify the error.

c. **Memory Checkerboard Tests.** These tests provide a "worst-case" data storage test which is unique to the memory module design of the applicable HP computer. These tests also consist of a Low-Core Test and a High-Core Test, verify that data is correctly stored in memory and is correctly transferred to and from the T-Register. Like the Memory Address Tests, the computer halts when an error is detected, and identifies the error on the front-panel display.

d. **Input/Output Tests.** A separate test program is supplied for most input/output devices in a user's hardware system. For example, the HP 2752A Teleprinter Test Program checks operation of the print, punch, and read functions with the computer. After it is determined that the print function is operating correctly, the program prints requests for data to be typed in so that the punch and read functions can be checked. Errors are indicated by a print-out. (Test programs for other devices require that a message printing facility, such as provided by the HP 2752A Teleprinter, be present in the hardware system.)



This chapter describes how an HP computer manipulates information internally to execute the basic instructions defined in the preceding chapter. Condensed descriptions of computer structure and implementation of instructions is provided to describe the mode of operation of all three Hewlett-Packard computers.

The fundamental operations described in this chapter are in practice nearly always accomplished with the aid of software and input/output devices. However, for simplicity it will be assumed that the computer is an independent instrument and will be operated only by front panel controls. Additionally, it will be assumed for descriptive purposes that the computer runs slowly enough to observe the operations step by step. When running, the computer reads and executes each instruction usually in one or two phases. Thus only the beginning and ending conditions are normally readable on the front panel display. (Note: It is possible to single-step the computer through each instruction, one phase at a time, by using the SINGLE CYCLE switch on the front panel.)

3.1 HP COMPUTER STRUCTURE

Figure 3.1, the Simplified Block Diagram of an HP computer, is the basis for the partial versions used to illustrate descriptions in this chapter. This figure will be reconstructed step by step as the explanations progress. The first step is Figure 3.2, which outlines the blocks and signal routes mentioned in the following discussion of memory. The block diagrams make use of several "and" gate symbols in addition to circuit blocks. These gates can produce an output only when all inputs are present ("true"). For example (referring to Figure 3.1), data on the T Bus can enter the T-Register only if a Store signal is also present at the gate leading to the T-Register input. Since the Store signal is selective (although this is not indicated on the diagram), only this one gate is enabled, while the remaining four are disabled. Thus the data enters only the selected register.

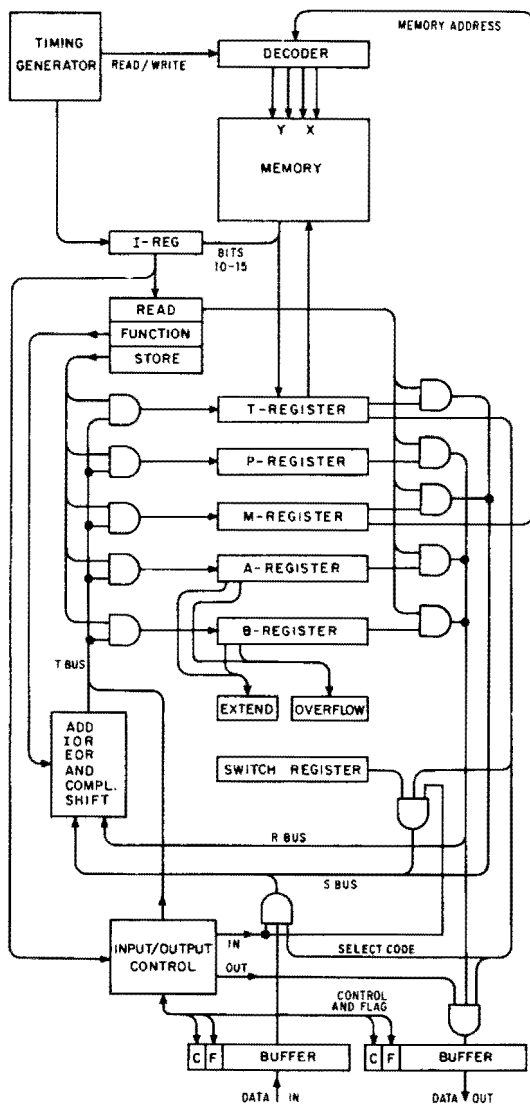


Figure 3.1. Simplified Block Diagram for Computers

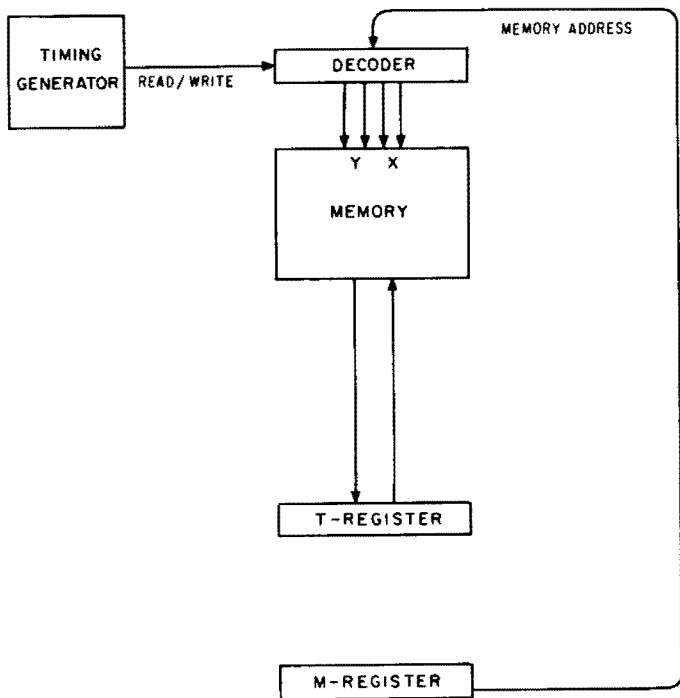


Figure 3.2. Memory Block Diagram

3.1.1 THE MEMORY MODULE

The primary storage of the HP computer is a "core memory", and is internal in the computer. (When more than two modules are installed in the 2116B only, the additional modules are housed in an external extender unit; however, memory access is the same as if all modules were inside the main frame.) Auxiliary storage is available in the form of disc storage (2116B/2115A) and magnetic tape; however, these units are accessed through the computer's input/output system and are not treated as an extension of memory in this discussion. Figure 3.3 shows the physical structure of the memory module, and the following paragraphs describe each of the four components identified in the figure, beginning with the smallest individual component, the ferrite core.

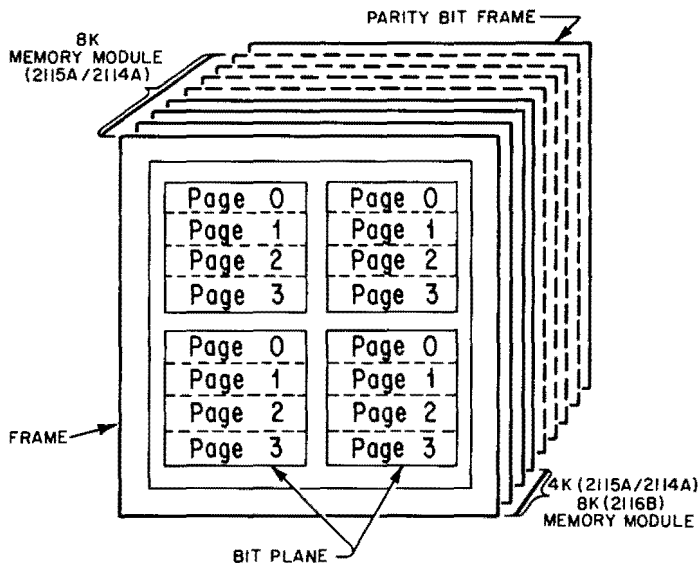


Figure 3.3. Core Memory Module

CORE. The computer handles all information in binary form; i.e., as a number representable by only two digits, zero and one. The ferrite core, which is a small ring of magnetic material, has the ability to store this binary information in that clockwise and counter-clockwise magnetization can be assigned digital values of one and zero. By threading a current-carrying wire through the core, the core's direction of magnetization can be reversed simply by changing direction of the current. Since the mass of the core is very small (diameter of .03 inch), little magnetizing force is required to switch the binary state, thus permitting fast switching speeds (about 400 nanoseconds). The magnetic state remains indefinitely after the current is removed, so that switching can be accomplished by bidirectional current pulses. This is shown in Figure 3.4.

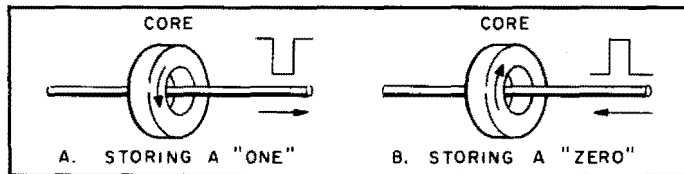


Figure 3.4. Binary Storage in a Magnetic Core

Since it is necessary to be able to select desired units of information in the module, four wires are required to be threaded through each core, as in Figure 3.5. In practice, the wires do not "loop" through the core, as shown for clarity in the figure, but simply pass through the center of a series of cores. Figure 3.5 shows how one bit of information is addressed and transferred to and from the T-Register. Action is as follows:

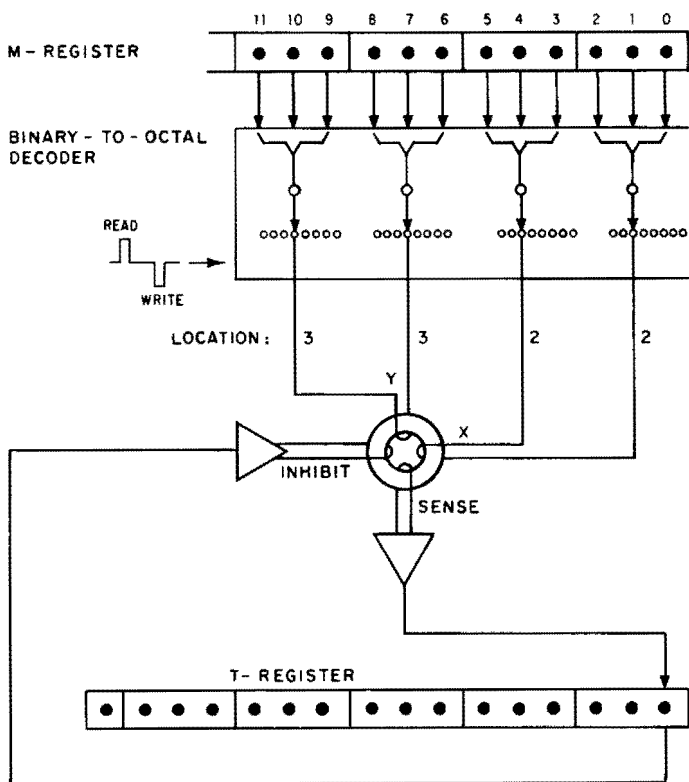


Figure 3.5. Core Addressing, Reading, and Writing

- a. Assume that the computer is running, and that the program has set the M-Register to a memory location number (address), desiring access to that location.

b. The address from the M-Register, consisting of 12 binary bits, is applied to a binary-to-octal decoder, which reduces the 12 binary address lines to four octal lines which thread, in pairs, through the selected core. For purposes of illustration, the diode decoding matrix is shown as four switches. Note that each of these switches can select one of eight ends of X and Y wires, thus making possible $8 \times 8 \times 8 \times 8 = 4096$ combinations to address 4096 core locations.

c. At a specific time in the computer's timing sequence (start of each memory cycle), all 16 bits of the T-Register are reset to zero.

d. A Read pulse is then applied to the decoder. Many cores will receive either Y-current or X-current pulses, neither of which alone is sufficient to switch the state of the core, but only one core out of 4096 on a plane (explained below) receives both Y-current and X-current pulses. The Read current is always in the direction which would magnetize the core in the "zero" direction. (If more than one module is present, module selection is accomplished simply by routing the Read pulse to the appropriate module, as determined by Bits 12, 13, 14 of the M-Register.)

e. If the core was previously magnetized in the "one" direction, the Read current, in switching the core, causes a flux change which induces a current into the Sense output line. This output is amplified and used to set the corresponding bit flip-flop of the T-Register (assumed as Bit 0 in Figure 3.5). If the core was in the "zero" state, there is no flux change and the T-Register bit remains zero (as reset in step "c").

f. Since steps "d" and "e" destroyed the stored information, it is necessary to "write" the information back. This information, which is now in the T-Register, is connected back to the core via the Inhibit line. Then the X and Y lines are pulsed with a Write current pulse, which is of opposite polarity to the Read pulse (i.e., tending to magnetize in the "one" direction).

g. If the inhibit current is not turned on, the core switches back to the "one" state. If the Inhibit current is turned on, it cancels part of the Write magnetizing force, so that the core cannot switch, and the core remains in the "zero" state.

The sequence of events in the preceding paragraph briefly describes the "memory cycle". There are two exceptions which modify the memory cycle slightly: 1) during the Execute phase of the "store" instructions (STA, STB, JSB), the output of the Sense Amplifier is inhibited and, instead, the data to be stored is transferred into the

T-Register from the A or B Register during the Read time period;
 2) during the Execute phase of the ISZ instruction (Increment, Skip if Zero), the T-Register is incremented between the Read and Write time periods.

MEMORY LOCATION. The word length of the HP computer is 16 bits, only one of which is shown in Figure 3.5. To store one 16-bit word, 16 cores are required, as indicated in Figure 3.3. These 16 cores comprise a "memory location", sometimes also referred to as a "memory cell". When information is transferred into or out of a memory location, the information in all 16 bits must be transferred simultaneously. Therefore the X and Y selection lines will be strung through the 16 cores, causing reading and writing of all 16 cores simultaneously. Figure 3.6 illustrates this, showing only three cores for simplicity. Note that each of these cores is on a different "plane" (next defined).

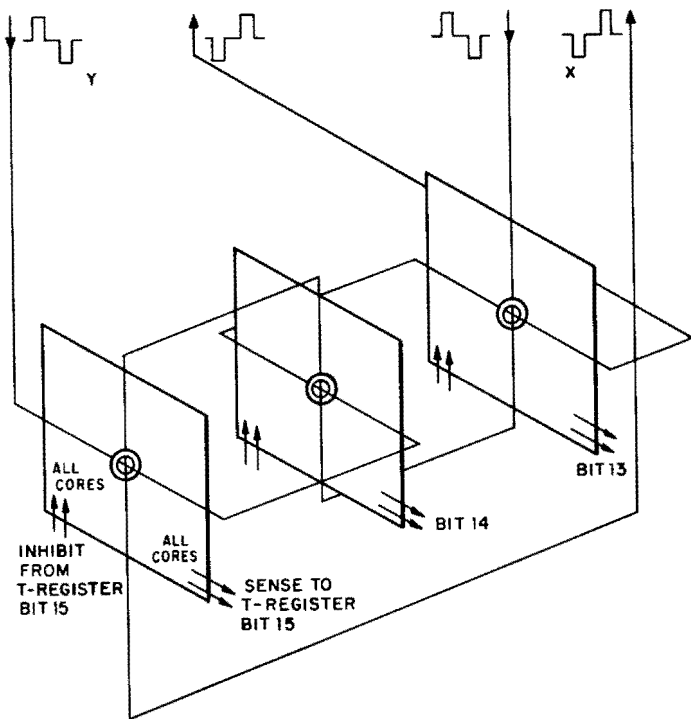


Figure 3.6. Memory Cell Selection

BIT PLANE. Cores are strung on a grid of wires, called a bit plane, and there are four of these bit planes on a frame for 2115A/2114A Computers. One of the bit planes is shown in Figure 3.7. The memory module for a 2116B has frames containing 8 bit planes, four on each side. Thus, the memory modules for the 2115A/2114A with 4K memory and the 2116B with 8K memory have the same number of frames. Both have five frames, with the fifth frame containing a single bit plane for the parity bit in 4K modules, and two bit planes for parity in 8K modules. To expand the memory capability of a 2115A or 2114A Computer from 4K to 8K requires installation of an 8K module in place of the 4K module. The 8K module contains nine frames; the first eight containing 4 bit planes each and the ninth frame containing two parity bit planes. The 2116B memory is expanded in 8K increments only and the maximum capability of 16K in the mainframe is provided by the addition of another 8K module. Each bit position of the T-Register is wired by the Sense and Inhibit lines through all cores on the corresponding bit plane. Since only one core on an individual plane is sensed (addressed) at a given instant of time, the Sense line needs only to detect a flux change anywhere on the bit plane. Similarly, the Inhibit signal is applied to the entire bit plane when writing, but actually affects only the selected core.

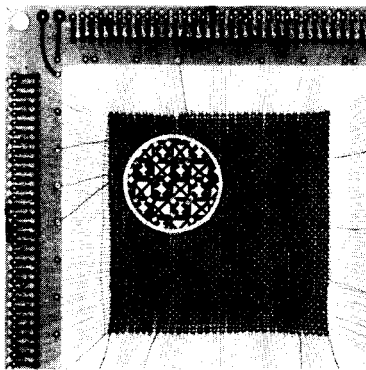


Figure 3.7. A Bit Plane and Frame
(Upper Left Corner)

PAGE. Pages of memory are not physical divisions of the module. Wiring of the bit planes is symmetrical and does not account for page boundaries. The page boundaries are determined only by the bit format of Memory Reference instructions, and are shown as broken lines in Figure 3.3 for visualizing the physical placement of memory pages.

3.1.2 THE REGISTERS

Figure 3.8 shows the seven working registers of an HP computer. The five principal registers (T, P, M, A, B) are purposely shown as being independent of each other since, in fact, information is not transferred directly from register to register. Rather, information is transmitted via the Bus System (described later under Paragraph 3.1.3) under command of the Instruction Logic (Paragraph 3.1.4). The following paragraphs explain why the registers are needed, not how they are operated. In essence, these registers are short-term information storage devices consisting of flip-flop circuits, with front-panel indicator lamps to indicate the status of each bit (on the 2114A, only the T- and M-Registers are displayed).

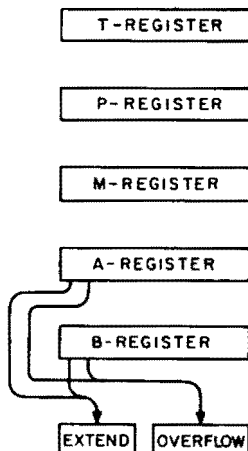


Figure 3.8. Register Block Diagram

T-REGISTER. As can be assumed from the front-panel engraving (MEMORY DATA), the T-Register holds data that is read out of and written into memory. For the majority of operations when a computer is running, the principal concern is with the data read out of a memory cell; once a word of information is in the T-Register, it is accessible for arithmetic operations and for transfers to other registers via the Bus System. For the reverse (Write) operation, the T-Register is loaded by transfers from other registers, and the information is stored in memory during the latter half of the memory cycle.

P-REGISTER. The P-Register is the computer's Program Counter. This means that this register goes through a step-by-step counting sequence and causes the computer to read successive memory locations, corresponding to the existing count. In the simplest case, The P-Register would start at zero when the RUN pushbutton is pressed, causing memory location 00000 to be read into the T-Register; the computer would act on the instruction code in the read-out data, then advance the P-Register to one (memory location 00001). This process of stepping through memory locations (at a rate of one or two machine phases per step for most instructions) continues until one of the instructions read out is a halt, which terminates the program. Of necessity, this simple case is not typical. First, programs do not normally begin at locations lower than 00077 , since these locations are reserved for special purposes. Therefore the starting address of a program must be manually set into the P-Register before pressing RUN. Second, the strict sequential stepping can be altered in the course of a program, either by a skip instruction (which causes the P-Register to increment by two instead of one, thus skipping one memory location) or by a jump instruction (which transfers numbers from another register into the P-Register, thus causing the program to continue at a different point in memory).

M-REGISTER. The M-Register (MEMORY ADDRESS) is the only means of addressing specific memory locations. The setting of the M-Register can occur from any of the other registers, depending on the effects of instructions. In the preceding paragraph, it could be assumed that the P-Register directly addresses memory; in actual fact, however, the computer must transfer the desired address from the P-Register to the M-Register, which in turn addresses the desired memory location. Thus it is seen that these two registers will frequently contain the same number. The reason why both registers are needed is that it is necessary for one register (the P-Register) to keep track of the location of the current instruction in case the instruction is a multiple phase type. In this case, the M-Register may have to be changed several times in the course of executing an instruction. A common example would be

when the instruction is to "add the contents of location 100₈ to the A-Register" (ADA 100). The P and M Registers would be identical while reading this instruction out of memory (say the instruction is in location 500₈; both registers indicate this value). Then the M-Register would have to change to 100 to get the contents of this location for the addition. After the addition has been executed, the contents of the P-Register are incremented by one (501₈). The P and M Registers are then both set to this new value, and the computer is ready to read the next instruction.

A-REGISTER. The A-Register is one of the computer's two accumulators. An accumulator in a computer accumulates the results of arithmetic operations. A simple example was given in the preceding paragraph, where one number from memory was added to the existing contents of the A-Register. Assuming that the A-Register previously held the number 1000₈, and the number in location 100 was 22₈, the number left in the A-Register after execution of the instruction would be 1022₈. Other types of operations which may be done with the A-Register are: boolean logic operations ("and", "exclusive or", "inclusive or"), comparison for equality with a memory word, shifting or rotating of bits left or right, testing the status of individual bits, complementing of bits, and accepting or holding data for transfer to and from external devices. All of these operations are accomplished by the Instruction Logic (Paragraph 3.1.4).

B-REGISTER. The B-Register is the second of the two accumulators. It has the same capabilities as the A-Register, except that the three boolean logic instructions (AND, XOR, IOR) can apply only to the A-Register. The main reason for having two accumulators is to provide faster, more flexible arithmetic than can be accomplished with one accumulator. This advantage will be seen later in programming of the computer.

EXTEND. The Extend register is shown connected to Bit 15 (left end bit) of both A and B Registers in Figure 3.8. This is to indicate that this one-bit register becomes set whenever there is a carry out of Bit 15 of either accumulator; i.e., whenever the quantity accumulated exceeds 16 ones. This fact is frequently of significance. For example, if the quantity in an accumulator is 16 ones and an ADD instruction adds one, the result in the accumulator will be 16 zeros. This answer is obviously incorrect; it is correct if the Extend bit, which is now in the set state ("1") is temporarily assumed to be "Bit 16". The program can be written to make this assumption, and it can proceed without error on the basis of the resulting information. To be certain that the Extend information is valid, the Extend register is normally cleared by an instruction (CLE) before the addition is done. Another valuable

feature of the Extend register, is its ability to link the two accumulators (effectively providing a single 32-bit accumulator).

OVERFLOW. The Overflow register is similar in purpose to the Extend register. The difference is that, whereas the Extend register indicates that the largest 16-bit quantity has been exceeded, the Overflow register indicates that the largest "signed" quantity has been exceeded. (A program may work with both signed and unsigned numbers.) Since Bit 15 is the sign bit, Bit 14 (as shown in Figure 3.8) is the source of the significant carry. Having two possible signs (+ and -) means that detection of overflow requires two different sets of conditions. For addition of two positive numbers, overflow occurs if there is a carry from Bit 14 to Bit 15 in one of the accumulators. For addition of two negative numbers (which are represented in two's complement form), overflow occurs if there is not a carry from Bit 14 to Bit 15. Obviously overflow cannot occur when adding numbers of opposing signs, since the resulting quantity cannot be greater than the larger of the two numbers. As with the Extend register, the Overflow register should be cleared before an addition.

3.1.3 THE BUS SYSTEM

Figure 3.9 outlines the routes by which data travels internally from one register to another. Although the buses are represented by a single line in this figure, assume each line to be composed of 16 individual lines, one for each register bit. Included in the figure is an "Arithmetic Logic" block, which has not previously been discussed. It is shown here mainly to illustrate the linkage between buses.

The HP computer uses an "R-S-T" bus configuration. This is a conventional notation designating a three-bus system which applies two input buses (R and S) to an arithmetic unit with output on the third bus (T). The use of two input buses permits arithmetic operations combining the contents of two registers. A common example would be the execution of the "ADA 100" instruction previously used. In that example, the contents of location 100 is the number 22. During execution of the instruction, this number (22) would be read into the T-Register. The other number (1000₈) is in the A-Register. Simultaneously (by a method described under the next paragraph heading, Instruction Logic) both the T-Register and the A-Register are read onto their respective buses (S and R). The two numbers are added in the Arithmetic Logic circuits, and the result (1022₈) is stored via the T Bus back into the A-Register as the accumulated sum.

Note that several register combinations are possible as inputs to the Arithmetic Logic. One point worth noting is that since the A and B Registers are addressable as memory locations, the contents of these registers can be transferred via the R and T Buses into the T-Register. From this point, the contents can be combined in the manner described above with either accumulator (including combining the number with itself; e. g., "add A to A"). This is all accomplished in one instruction.

3.1.4 THE INSTRUCTION LOGIC

Figure 3.10 shows the elements of the Instruction Logic in the computer. As indicated in the figure, timing is essential to the operation of the Instruction Logic. The following descriptions do not detail all timing relationships, since these vary with instructions, but it should be understood that timing pulses are gated with each operation to make it occur in proper sequence.

As shown in Figure 3.10, the six most significant bits read out of memory during each memory cycle are applied to the 6-bit Instruction Register (I-Reg), which decodes the instruction. (Actually, the Instruction Register receives its information via the T-Register; for simplicity Figure 3.10 shows a direct connection to memory.) Only during the Fetch phase, however, are these bits recognized as an instruction code (as determined by a "Fetch Phase" signal from the Timing Generator). At this time, the decoded instruction enables three functional operations, which in turn will become active at specific times, depending on the instruction. These operations are described individually in the next three paragraphs.

READ. The Read signal, shown connected to the output gate of all five working registers, strobes the data of one or two registers onto their corresponding buses (R and S). This places the data at the inputs of the arithmetic logic circuits

FUNCTION. The Function signal activates one of the six listed arithmetic functions. The selected function alters or combines the data on the R and/or S Buses, and routes the resulting data out on the T Bus.

STORE. The Store signal, shown connected to the input gate of all five working registers, effectively opens the input of one or more of these registers to accept the data which appears on the T Bus (preceding paragraph). In many cases, depending on the instruction, only part of the information on the T Bus is stored into a register.

3.1.5 THE INPUT/OUTPUT SYSTEM

Figure 3.11 shows the means by which data is transferred in and out of the computer. This is the Input/Output System; all elements shown are contained within the mainframe. Interface arrangements are shown for only two external devices, one input and one output. The Switch Register is shown as part of the Input/Output System, and is considered to be an input device.

As indicated by Figure 3.11, the Input/Output Control logic is used to process all input/output operations. Input/Output Control operates in two ways:

- a. Processes input/output instructions.
- b. Processes service requests by peripheral devices.

These two types of operations are separately discussed in the following paragraphs.

PROCESSING INSTRUCTIONS. Input/Output instructions decoded by the Instruction Register are routed to Input/Output Control, which translates the instruction into appropriate driving signals. One such signal is an "In" signal, which strobes all interface positions for input (represented by two "and" gates in Figure 3.11, one accepting data from a Buffer register and one accepting data from the Switch Register). Only one of these interface positions can be enabled, according to the Select Code (Bits 0 through 5 from the T-Register), and the corresponding data is strobed by the "In" pulse onto the S Bus. From there it is transferred via the T Bus into the A or B Register (as enabled by a Store signal at the A or B input gate).

Another driving signal is the "Out" signal. This signal strobes all interface positions for output (one shown in Figure 3.11). The Select Code from the T-Register enables one interface position, and permits the "Out" signal to strobe the data on the R Bus into the corresponding output Buffer. (The data on the R Bus was read out of the A or B Registers by a Read signal.)

In addition to transferring data, as in the preceding two paragraphs, Input/Output Control can (according to instruction) send out signals to test the state of Control and Flag bits (C and F), or to set or reset these bits. The Select Code determines which interface will receive the signal from Input/Output Control. The Control and Flag bits are command signals for transferring data between the Buffer and the peripheral device (peripheral not shown).

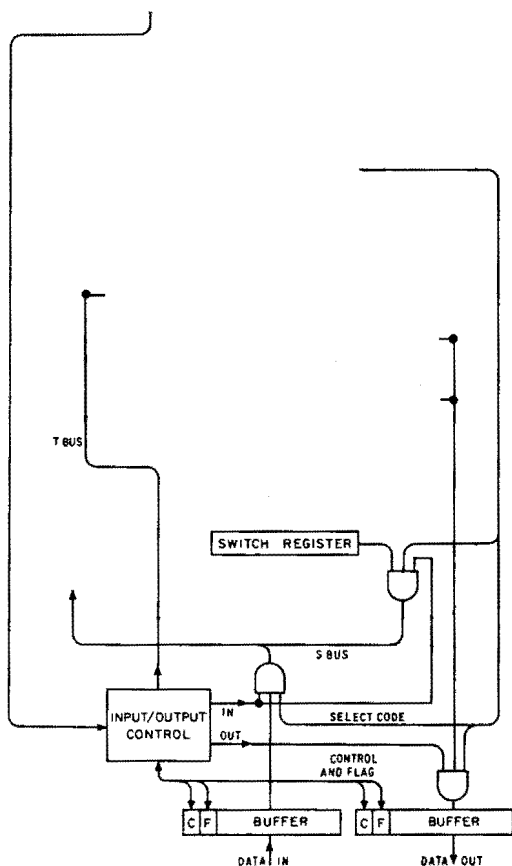


Figure 3.11. Input/Output System Block Diagram

PROCESSING SERVICE REQUESTS. If a specific instruction has at some previous time enabled the interrupt system (considered to be in the Input/Output Control block in Figure 3.11), a peripheral device may request new data from the computer (if output) or request to feed new data to the computer (if input). This request for service is done by setting the interface Flag bit. The Flag signal, via Input/Output Control, interrupts the computer's operation by forcing the M-Register to be set (via the T Bus) to a memory address uniquely specified by the Flag. At the same time, the Fetch phase is set so that the computer must execute the instruction contained in the specified memory cell. Generally this instruction will be a jump to a service subroutine. This subroutine consists of instructions that will prepare or accept the new data. On completion of service, it is the subroutine's responsibility to return the P and M Registers to the values they contained before being interrupted.

3.2 IMPLEMENTATION OF INSTRUCTIONS

The following paragraphs describe how the 70 basic instructions are implemented internally in the computer. Figures 3.12 through 3.14 expand on the Machine Timing diagram (Figure 2.2) given in Chapter 2. Figure 3.1, the Simplified Block Diagram, is also used as a reference throughout the following descriptions. Most signals named can be identified in this figure; e.g., "Read A onto R Bus" is the line from the Read block to the A-Register output gate (which outputs onto the R Bus). The block diagram should be referred to frequently as the discussion progresses, in order to visualize the bit manipulations. The right-pointing arrows in the figures should be read as "into" or "onto" (e.g., "into" T-Register, or "onto" R Bus). New mnemonics are introduced in these descriptions which will be defined within the text.

The cycle of Time Periods shown at the top of Figures 3.12 through 3.14 (T0 through T7) repeats continuously every 1.6 or 2.0 microseconds, as applicable, while computer power is on. The Read/Write memory cycle, although shown only once at the top of each of these figures, actually occurs once in every phase (except Interrupt). It is important to remember this throughout the following descriptions.

3.2.1 MEMORY REFERENCE

By comparing Figures 3.12 through 3.14, it is seen that Memory Reference instructions are the only type of instructions requiring more than one machine phase to execute; Indirect and Execute phases are associated only with Memory Reference instructions.

In the case of all these instructions except JMP, the action during the Fetch and Indirect phases (Phases 1 and 2) is similar, so these phases are shown only once, implying that they are common to all Memory Reference instructions. The exception, JMP, is unique in that it does not use an Execute phase; execution can occur in either the Fetch or the Indirect phase. The action for JMP is shown separately in Figure 3.12 and is discussed first below.



Note

The descriptions for JMP and AND instructions are more detailed than for succeeding instructions, which are similar in many respects. These two should therefore be studied in detail before advancing to the others.

JMP. The Fetch phase for all instructions, regardless of type, begins in exactly the same way, since at this time the computer logic cannot know anything about the instruction which is about to be read out of memory. The only fact known is that the word from memory will be read as an instruction (not data); getting an instruction from memory is the first function of the Fetch phase. During the first three Time Periods of the Fetch phase, the following actions occur:

- a. During T0 the T-Register is cleared.
- b. The Read portion of the Memory Cycle begins to read the contents of the currently addressed memory cell into the T-Register. This continues until the middle of T2.
- c. During T1 the Instruction Register is cleared.
- d. Bits 10 through 15 (the instruction group and code identification) of the T-Register are transferred into the 6-bit Instruction Register.



During the latter portion of T2, the functions to be used in implementing the JMP instruction are set up. This includes Read and Store as well as any arithmetic functions (none in the case of JMP). Functions are gated with Time Periods to occur in the correct sequence.

PHASE		TIME PERIODS							
		T0	T1	T2	T3	T4	T5	T6	T7
		2116B: .2 μ Sec	.4	.6	.8	1.0	1.2	1.4	1.6
		2115A/2114A: .25	.5	.75	1.0	1.25	1.5	1.75	2.0
		READ (Mem to TR)				WRITE (TR to Mem)			
									
FETCH (JMP)	1	Clear TR	Clear IR	TR(10-15) - IR (Set Functions)			If Z: 0 - P	If Z: 0 - M (10-15) If D: TR -P, M (0-9) and set PH1 If I: TR -M (0-9) and set PH2	
INDIRECT (JMP)	2	Clear TR						If D: TR -P, M and set PH1 If I: TR -M and set PH2	
FETCH	1	Clear TR	Clear IR	TR(10-15) - IR (Set Functions)				TR -M (0-9) If Z: 0 - M (10-15) If I: Set PH2 If D: Set PH3	
INDIRECT	2	Clear TR						TR - M If I: Set PH2 If D: Set PH3	
EXECUTE AND	3	Clear TR			Read A -R Bus Read TR -S Bus Store T Bus (ANF) -A			Read P -R Bus Read "1" -S Bus Store T Bus (ADF) - P, M Set PH1	

XOR IOR JSB ISZ ADA/B CPA/B LDA/B STA/B	Clear TR		A (EOF) TR \rightarrow A			P + 1 \rightarrow P, M Set PH1
	Clear TR		A (IOF) TR \rightarrow A			P + 1 \rightarrow P, M Set PH1
	Clear TR Inhibit Mem. Data	P + 1 \rightarrow TR	M \rightarrow P			P + 1 \rightarrow P, M Set PH1
	Clear TR		TR + 1 \rightarrow TR If C16: Set Carry Inhibit Write		Write*	P + 1 + Carry \rightarrow P, M Set PH1
	Clear TR		If A: A (ADF) TR \rightarrow A If B: B (ADF) TR \rightarrow B If C16: Set E			P + 1 \rightarrow P, M Set PH1
	Clear TR		If A: A (EOF) TR \rightarrow T Bus If B: B (EOF) TR \rightarrow T Bus If T Bus not zero, set Carry			P + 1 + Carry \rightarrow P, M Set PH1
	Clear TR		If A: TR \rightarrow A If B: TR \rightarrow B			P + 1 \rightarrow P, M Set PH1
	Clear TR Inhibit Mem. Data	If A: A \rightarrow TR If B: B \rightarrow TR				P + 1 \rightarrow P, M Set PH1



* 2116B: Add $0.4\mu\text{S}$ 2114A/2115A: Add $0.5\mu\text{S}$

Figure 3.12. Implementing Memory Reference Instructions

		TIME PERIODS							
		T0	T1	T2	T3	T4	T5	T6	T7
		2116B: .2 μ Sec	.4	.6	.8	1.0	1.2	1.4	1.6
		2115A/2114A: .25	.5	.75	1.0	1.25	1.5	1.75	2.0
		READ (Mem to TR)			WRITE (TR to Mem)				
									
FETCH	1	Clear TR	Clear IR	TR (10-15) - IR	— Execute —		P +1 +Carry - P, M Set PH1		
SHIFT-ROTATE INSTRUCTIONS		T3			T4		T5		
		All Shifts and Rotates Read A or B \rightarrow R Bus Shift R Bus \rightarrow T Bus Store T Bus \rightarrow A or B			Clear E and Skips If TR5 = 1 - CLE If TR3 = 1 (SLA/B): Read A or B \rightarrow R Bus If RB0=0 - Set Carry		All Shifts and Rotates Read A or B \rightarrow R Bus Shift R Bus \rightarrow T Bus Store T Bus \rightarrow A or B		
		CLA/B: No Read (R Bus all zeros) Store T Bus (EOF) \rightarrow A/B			*SSA/B: Read A/B \rightarrow R Bus Set Carry if RB15 = 0 and TR0 = 0, or RB15 = 1 and TR0 = 1		SZA/B: Read A/B \rightarrow R Bus (ADF) \rightarrow T Bus Set Carry if T Bus all zeros and TR0 = 0, or if T Bus all ones and TR0 = 1		
		CMA/B: Read A/B \rightarrow R Bus Store T Bus (CMF) \rightarrow A/B			*SLA/B: Read A/B \rightarrow R Bus Set Carry if RB0 = 0 and TR0 = 0, or RB0 = 1 and TR0 = 1				
ALTER-SKIP INSTRUCTIONS									

	CCA/B: No Read (R Bus all zeros) Store T Bus (CMF) - A/B	INA/B: Read A/B -R Bus Read "1" -S Bus Store T Bus (ADF) - A/B If C16: Set E	
	SEZ: Set Carry if E = 0 and TR0 = 0, or E = 1 and TR0 = 1		
	CLE: Reset E Flip-flop		
	CME: Complement E Flip-flop		
	CCE: Set E Flip-flop		* Combination of SSA/B, SLA/B, and RSS is a special case; see text.

Figure 3.13. Implementing Register Reference Instructions

PHASE		TIME PERIODS							
		T0	T1	T2	T3	T4	T5	T6	T7
		2116B: .2 μ Sec	.4	.6	.8	1.0	1.2	1.4	1.6
		2115A/2114A: .25	.5	.75	1.0	1.25	1.5	1.75	2.0
		READ (Mem to TR)			WRITE (TR to Mem)				
									
FETCH	1								
	HLT	Clear TR	Clear IR	TR(10-15) - IR				P +1 - P, M Reset Run FF	
	STF	Clear TR	Clear IR	TR - IR	Set Flag: Select Code			P +1 - P, M Set PH1	
	CLF	Clear TR	Clear IR	TR - IR	Set Flag: Select Code	Clear Flag: Select Code			P +1 - P, M Set PH1
	SFC	Clear TR	Clear IR	TR - IR	SFC - Interface	SKF - Carry			P +1 +Carry - P, M Set PH1
	SFS	Clear TR	Clear IR	IR - IR	SFS - Interface	SKF - Carry			P +1 +Carry - P, M Set PH1
	MIA/B	Clear TR	Clear IR	TR - IR			Read A/B - R Bus Buffer - S Bus Store T Bus (IOF) - A/B		P +1 - P, M Set PH1
						TR9: CLF			

LIA/B OTA/B STC CLC STO CLO SOC SOS		Clear TR	Clear IR	TR - IR		Buffer → S Bus Store T Bus (IOF) → A/B TR9: CLF	P+1 - P, M Set PH1
		Clear TR	Clear IR	TR - IR		Read A/B - R Bus R Bus → Buffer TR9: CLF	P+1 - P, M Set PH1
		Clear TR	Clear IR	TR - IR		Set Control (Sel. Code)	P+1 - P, M Set PH1
		Clear TR	Clear IR	TR - IR		Clr. Control (Sel. Code)	P+1 - P, M Set PH1
		Clear TR	Clear IR	TR - IR	STF - Overflow		P+1 - P, M Set PH1
		Clear TR	Clear IR	TR - IR		CLF - Overflow	P+1 - P, M Set PH1
		Clear TR	Clear IR	TR - IR	SFC - OVF	SKF Carry	P+1+Carry - P, M Set PH1
		Clear TR	Clear IR	TR - IR	SFS - OVF	SKF - Carry	P+1+Carry - P, M Set PH1
INTERRUPT	4		Read P - R Bus Store T Bus (CMF) - P	Read P - R Bus Read "1" - S Bus Store T Bus (ADF) - P	Read P - R Bus Store T Bus (CMF) - P,		Reset M (6-15) Store T Bus (0-5) - M Set PH1

Figure 3.14. Implementing Input/Output Instructions

At this point in time (end of T2), the instruction information is in Bits 10 through 15 of the T-Register, and in the Instruction Register. The Memory Address information is in Bits 0 through 9 of the T-Register. The next event to occur is to clear the P-Register at time T5 if the page Zero condition exists (i.e., if Bit 10 of the Instruction Register is a zero). This is done by a "Store T Bus into P" function. Since nothing has been read onto any of the buses, the T Bus is in the all-zero state, and 16 zeros are therefore stored into the P-Register. (Actually, for resetting the program to page Zero, it is only necessary to clear Bits 10 through 14 of the P-Register; however it is convenient to clear the entire P-Register at this time.) Note that the 6 most significant bits of the page Zero address are zeros; e.g., the last address on page Zero is:

0 000 001 111 111 111

During Time Periods T6 and T7, the page Zero indicator (if present) clears Bits 10 through 15 of the M-Register (not the entire register). The method is the same as described above: "Store T Bus into M-Register, Bits 10 through 15"; the T Bus is still all zeros. Thus, at this time both P and M Registers point to page Zero, if so coded by Bit 10 being a zero (otherwise these registers are not changed, leaving Bits 10 through 15 at the Current page indication).

Also during T6 and T7, the Direct/Indirect bit (Bit 15) of the T-Register is looked at, to see if the Memory Address currently in the T-Register is the "effective address" (the final address being jumped to), or if another jump should be made from that address to whatever address is contained in that location (indirect addressing). Since the concept of indirect addressing is important and not always simple to grasp initially, it is treated separately in following paragraphs. For direct addressing, the execution is completed by the following steps:

- a. The T-Register contents are read onto the S Bus, and appear on the T Bus.
- b. Bits 0 through 9 of the T Bus are stored into the P and M Registers. This directs the Computer to the "jump" location. (Remember from the preceding paragraphs that Bits 10 through 15 of the P and M Registers either have been reset to zero for page Zero or have been left alone for Current page.)
- c. The Phase 1 (Fetch) condition remains set so that the contents of the "jump" location will be read out and interpreted as an instruction during the next machine phase.

Basically, the Indirect addressing indicator (Bit 15 of T-Register being a one) tells the computer logic that the contents of the location

being jumped to is not the next instruction, but rather the address for another jump. This additional jump is a continuation of the same instruction, but requires an additional phase. During T6 and T7 of Phase 1, the T-Register contents are transferred to the M-Register (not both P and M as for the Direct condition). During T7 the Phase 2 condition (PH2) is set, and the Indirect phase begins.

During T0, the T-Register is cleared. Since the "jump" is still in progress, the Instruction Register is not cleared during T1. The contents of the location now addressed by the M-Register are read into the T-Register during the Read memory cycle. Then, during T6 and T7 (assuming Bit 15 of the T-Register is now 0 for Direct), all 16 bits of the T-Register are transferred into the P and M Registers in the usual way: read T-Register onto S Bus, and store T Bus (with no arithmetic) into P and M Registers. These registers now contain the effective address, so Phase 1 is set, and the next machine phase will be a Fetch phase, to read out the next instruction from that address. Note that if Bit 15 of the T-Register were again a one (for Indirect) a jump would be made to still another location by repeating the process.

In summary, as illustrated in Figure 3.15, an indirect jump occurs by the following register actions:

- a. The word containing the jump instruction is read out of memory by a Fetch phase into the T-Register.
- b. The address portion of the read-out word is transferred into the corresponding portion of the M-Register.
- c. The Zero/Current page bit of the read-out word tells the computer logic to clear (Zero) or leave (Current) the remaining bits (10 through 15) of the M-Register.
- d. Steps "b" and "c" now comprise the address of a location which is read out of memory into the T-Register at the start of the Indirect phase.
- e. All bits of this new read-out word are transferred into the P and M Registers. The computer is now "at" the location specified by these Registers.

AND. The Fetch phase for the AND instruction is the same as for all other Memory Reference instructions listed below it in Figure 3.12, with the exception that different functions will be set up at T2. This phase begins in the same way as for JMP: the T-Register is cleared at time T0, the Read memory cycle reads the instruction word into the T-Register, the Instruction Register is cleared

during T1, and T-Register Bits 10 through 15 (instruction code) are transferred into the Instruction Register at T2. At this time all necessary functions for this instruction are set up, to be used at the appropriate times. During T6 and T7, T-Register Bits 0 through 9 (memory address portion of the instruction word) are transferred into the corresponding bits of the M-Register (via S and T Buses). If the Zero page indicator is present (Bit 10 of the Instruction Register is a zero), a "Reset M(10-15)" command clears Bits 10 through 15 of the M-Register.

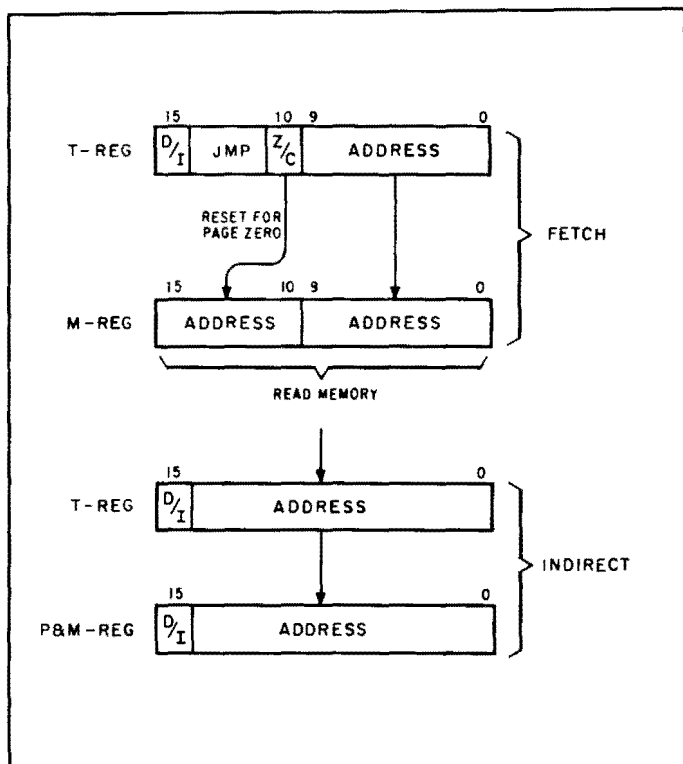


Figure 3.15. Register Manipulations for Indirect Jump

Unlike the JMP instruction, an Execute or an Indirect phase must follow the Fetch phase of an AND instruction. (Execute never occurs for JMP; Indirect is optional.) If Bit 15 of the T-Register is zero (for Direct), Phase 3 (Execute) is set. Assume an Indirect phase is required (Bit 15 = 1). (If the Direct condition exists, the action of the next paragraph would be skipped.)

The Indirect phase begins by clearing the T-Register during T0. Then a new word is read into the T-Register from the memory location specified by the M-Register. This word is an address, not data, since indirect addressing really means: "go to another location for the data". During T6 and T7 of the Indirect phase, this address is transferred from the T-Register to the M-Register (all 16 bits). Note that it is possible for Bit 15 to again specify Indirect addressing; if so, Phase 2 remains set and the procedure of this paragraph is repeated, and could be repeated several times. When Bit 15 is a zero (Direct), Phase 3 is set.

The Execute phase begins by clearing the T-Register. The Instruction Register remains unchanged, since the various functions are still needed. This time, the Read portion of the memory cycle reads data from memory into the T-Register. During T3 and T4, this data is read onto the S Bus and the A-Register contents are read onto the R Bus. The "and" function (ANF) previously set up by the Instruction Register, now combines the data on the two buses by "anding". The result on the T Bus is then stored into the A-Register.

To advance the computer to the next instruction, the P and M Registers must be incremented by one. This is done during T6 and T7 of the Execute phase. It is accomplished by reading the P-Register onto the R Bus and a "one" onto the S Bus, then adding the two buses (Add Function: ADF) and storing the result into the P and M Registers.

In summary, as illustrated in Figure 3.16, an AND Indirect instruction is executed by the following register actions:

- a. The word containing the AND instruction is read out of memory by a Fetch phase into the T-Register.
- b. The address portion of the read-out word is transferred into the corresponding portion of the M-Register.
- c. The Zero/Current page bit of the read-out word tells the computer logic to clear (Zero) or leave (Current) the remaining bits of the M-Register.

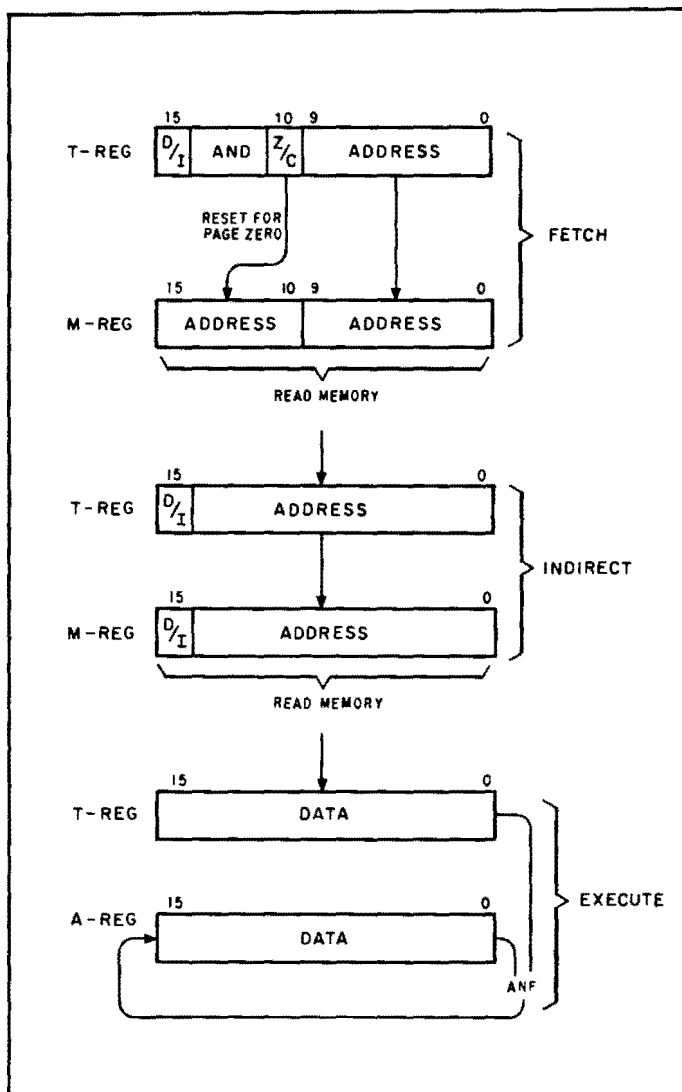


Figure 3.16. Register Manipulations for Indirect AND

d. Steps "b" and "c" now comprise the address of a location which is read out of memory into the T-Register at the start of the Indirect phase.

e. All bits of this new read-out word are transferred into the M-Register, thus addressing the location of the desired data.

f. At the start of the Execute phase the data thus addressed is read into the T-Register from memory.

g. The contents of the T-Register and A-Register are "anded" together and deposited back into the A-Register.

Note

For the remainder of Memory Reference instructions, the Fetch and Indirect phases are the same as described above for the AND instruction. The following paragraphs therefore describe only the Execute phase for each instruction.

XOR. The Execute phase of the XOR ("Exclusive Or") instruction begins as usual by clearing the T-Register just before the Read portion of the Memory cycle. The action occurring during T3 and T4 is shown in abbreviated form in Figure 3.12, to be read as follows: the contents of the A-Register is combined by an "exclusive or" function with the contents of the T-Register, and stored back into the A-Register. Actually this action consists of three steps as shown for the AND instruction. For XOR, these three steps are: 1) Read T-Register onto S Bus; 2) Read A onto R Bus; 3) Store T Bus (which carries the "exclusive or" combination of the S and R Buses) into the A-Register. The action during T6 and T7 is also abbreviated: add "one" to P, and store into P and M. The three steps which accomplish this are detailed for the AND instruction in Figure 3.12. The last action is to reset the computer to the Phase 1 (Fetch) condition.

IOR. The Execute phase of the IOR ("Inclusive Or") instruction is the same as XOR described in the preceding paragraph, except that the "inclusive or" function is used in place of "exclusive or". The difference in arithmetic is shown in Table 2.2 of the previous chapter.

JSB. The principal operation of the Execute phase for JSB (Jump to Subroutine) is to store the return address (Program Counter contents plus one) in the memory location being jumped to. This is done during T0 through T2. Since the only way into memory is through the T-Register, the T-Register must be loaded with the

return address prior to the Write portion of the memory cycle. Therefore the memory contents read out during the Read portion of the memory cycle must be inhibited and, instead, (during T1 and T2) the current contents of the P-Register, plus one, is stored into the T-Register. (Action: Read P onto R Bus, Read "1" onto S Bus, Store with add function into T-Register.) This information is then stored into memory during Write. To complete the jump process, the contents of the M-Register (which received the "jump" memory address during the Fetch or Indirect phase) must be transferred into the P-Register. This is done during T3: Read M onto S Bus, Store T Bus in P. As usual, to advance the computer to the location of the next instruction both P and M Registers are incremented by one during T6 and T7, and the Fetch phase condition is set.

ISZ. During the Execute phase of the ISZ instruction (Increment, Skip if Zero), the contents of the addressed memory cell must be altered and checked between the Read and Write portions of the memory cycle. These actions require more time than is normally available in this interval, so the Write portion is delayed. Once the word read from memory is in the T-Register (T3 and T4), it is incremented by reading onto the S Bus, adding "one" in the arithmetic logic and storing back into the T-Register. If previously the word read out was all ones, the addition of another one causes a rollover to all zeros, and produces a signal (C16) which sets a Carry flip-flop in the arithmetic logic. Then, at T5, the Write portion of the memory cycle is permitted to begin, and two Time Periods ($0.4\mu\text{s}$ for 2116B; $0.5\mu\text{s}$ for 2114A and 2115A) are inserted at this time for writing the incremented value back into memory. During T6 and T7, the P-Register is read onto the R Bus, and a "one" is read onto the S Bus. These are added together, and if the Carry flip-flop is set, another "one" is added and the result is stored in the P and M Registers. Thus, if the Carry flip-flop was set, the P and M Registers are incremented by two instead of one, skipping one memory location for the next Fetch phase. (The Carry flip-flop is automatically reset at the start of the next phase.)

ADA/B. If Bit 11 of the Instruction Register indicated A (zero), the contents of the A-Register are combined with the T-Register contents by the add function (ADF), and stored into the A-Register. Similar action involving the B-Register occurs during this time (T3 through T4) if Bit 11 of the Instruction Register is a one.

CPA/B. Depending on the status of Bit 11 of the Instruction Register, either the A-Register or the B-Register is combined with the T-Register contents by the "exclusive or" function. The result appears on the T Bus, but is not stored anywhere. Logic not shown in Figure 3.1 tests the T Bus for a non-zero condition which, if it exists, sets the Carry flip-flop. Then during T6 and T7 (as for ISZ),

the P and M Registers are incremented by either one (Carry not set) or two (Carry set).

LDA/B. During T3 and T4, the information read into the T-Register by the Read portion of the memory cycle is simply transferred to either the A or B Register via the S and T Buses.

STA/B. Like JSB, the STA/B instruction (Store A or B) deposits new information into a memory cell, with no concern for the existing memory contents. The memory data read out during the Read portion of the memory cycle is therefore inhibited while the A or B Register contents are read and stored into the T-Register (during T1 and T2). The Write portion of the memory cycle deposits this information into memory.

3.2.2 REGISTER REFERENCE INSTRUCTIONS

All Register Reference instructions, as shown by Figure 3.13, are fully executed in only one phase (Fetch). Actual execution is accomplished during Time Periods T3 through T5. Actions during the other Time Periods are similar to those previously described for Memory Reference instructions:

a. During Time Periods T0 through T2, the T-Register and Instruction Register are cleared, and Bits 10 through 15 of the instruction word read out of memory are transferred to the Instruction Register. Unlike Memory Reference, the Instruction Register does not set up functions, but rather it provides gating signals to identify the type (Register Reference) and group (Shift-Rotate, or Alter-Skip) of instructions. The remaining bits of the T-Register are used to execute the individual instructions by setting up the appropriate functions. Figures 2.6 and 2.7 define which bits encode each instruction.

b. During Time Periods T6 and T7, the P-Register is read onto the R Bus and a "one" is read onto the S Bus. If the Carry flip-flop has been set by a "skip" condition during T3 through T5, another "one" is added and the total (P-Register incremented by one or two) is stored into the P and M Registers. This advances the computer to the next instruction.

3.2.3 SHIFT-ROTATE INSTRUCTIONS

Figure 3.13 shows that shifts and rotates can be executed either during T3 or T5, or both. CLE (Clear Extend) or SLA/B (Skip if Least significant bit of A or B Registers is zero) can be executed only during T4. The shifts and rotates are executed simply by reading A or B Registers onto the R Bus, applying a "Shift Function"

to shift some or all of the bits to a different position on the T-Bus, then storing the T Bus back into the A or B Register. Since the Shift Function is the key to understanding how shifts and rotates occur, the following instruction descriptions concentrate on this aspect (CLE and SLA/B are described later). Table 3.1 is the main reference for these descriptions.

A/BLS. As shown by the Table 3.1 diagram for A/BLS (A or B Left Shift), the desired end result is to have Bits 0 through 13 shifted left one place, with Bit 15 unchanged and a zero moved into Bit 0. Assuming that Bits 6 through 9 of the T-Register dictate an A/BLS during T3, an SLM (Shift Left Magnitude) signal at this time is "anded" with each of the 14 R-Bus bits (0 through 13), with the output of each "and" gate appearing on the next higher T Bus line. The Function listed in Table 3.1 for this instruction (SLM RB(0-13)) is therefore to be read: Shift Left Magnitude "anded" with R Bus Bits 0 through 13. Bit 15 of the R Bus is routed directly out to Bit 15 of the T Bus. Since nothing has been placed onto Bit 0 of the T Bus, its state is "zero", and therefore no deliberate action is necessary to ensure storing a zero in Bit 0 of the A or B Register.

A/BRS. A Shift Right Magnitude "anded" with R Bus Bits 1 through 15 shifts these bits to Bits 0 through 14 of the T Bus. Bit 0 of the R Bus is not recognized, and Bit 15 (as well as moving onto Bit 14 of the T Bus) also is routed directly to Bit 15 of the T Bus.

RA/BL. To rotate A or B left, an SLM "anded" with R Bus Bits 0 through 13, together with a "Shift Left bit 14" to R Bus Bit 14, move Bits 0 through 14 to Bits 1 through 15 of the T Bus. Rotating Bit 15 of the R Bus around to Bit 0 of the T Bus is accomplished by "anding" RLL (Rotate Left to Least significant bit) with R Bus Bit 15; the "and" gate outputs to T Bus Bit 0.

RA/BR. A Shift Right Magnitude "anded" with R Bus Bits 1 through 15 shifts these bits to Bits 0 through 14 of the T Bus. An RRS (Rotate Right to Sign bit) "anded" with R Bus Bit 0 rotates this bit to Bit 15 of the T Bus.

A/BLR. A Shift Left Magnitude with R Bus Bits 0 through 13 shifts these bits to Bits 1 through 14 of the T Bus. Bits 0 and 14 of the T Bus remain in the "zero" state, since nothing is placed on these lines.

ERA/B. A Shift Right Magnitude with R Bus Bits 1 through 15 causes shift to T Bus Bits 0 through 14. The content of the Extend register is transferred into Bit 15 of the T Bus. Then, during the latter half of T3 (or T5), Bit 0 of the R Bus is transferred into the Extend register.

ELA/B. A Shift Left Magnitude "anded" with R Bus Bits 0 through 13, and a Shift Left 14 with R Bus Bit 14 shifts these bits to Bits 1 through 15 of the T Bus. The Extend content is transferred onto T Bus Bit 0, and then Bit 15 of the R Bus is transferred into the Extend register.

A/BLF. A Rotate Left 4 "anded" with all bits of the R Bus shifts each bit four places to the left on the T Bus. The four most significant bits are placed into the least significant bit positions.

CLE. During T4, if Bit 5 of the T-Register is a one, a reset signal is generated which clears the Extend register.

SLA/B. During T4, if Bit 3 of the T-Register is a one, the A or B Register is read onto the R Bus. (Bit 11 determines which register is read out.) If Bit 0, the least significant bit, is a zero, the Carry flip-flop is set. This will cause the P and M Registers to be incremented by two (for a skip) during T6 and T7.

3.2.4 ALTER-SKIP INSTRUCTIONS

Figure 3.13 individually lists all Alter-Skip instructions. The grouping into three Time Periods explains the grouping of columns in the Selection Table of Figure 2.7. That is, during T3 one instruction involving the accumulators can be executed (clear, complement, or clear-complement), and two possible instructions involving the Extend register can be executed (skip if zero, and clear or complement or clear-complement). Incrementing of accumulators (INA/B) effectively occurs after tests for sign and least significant bits (SSA/B and SLA/B, at T4), but before the test for zero accumulator (SZA/B, at T5).

The alter instructions (clear, complement, and increment) use a Store or direct transfer function. The skip instructions, however, simply read information onto the T Bus for testing; a Store function is not required. If skip conditions are met, the Carry flip-flop is set, causing the P and M Registers to be incremented by two during T6 and T7.

CLA/B. To clear the A or B Register, the Read function is omitted. This means that both R and S Buses are in the all-zero state. The "exclusive or" function, in combining zeros with zeros, can only produce zeros on the T Bus. Thus when the T Bus is stored into A or B, the result is all zeros.

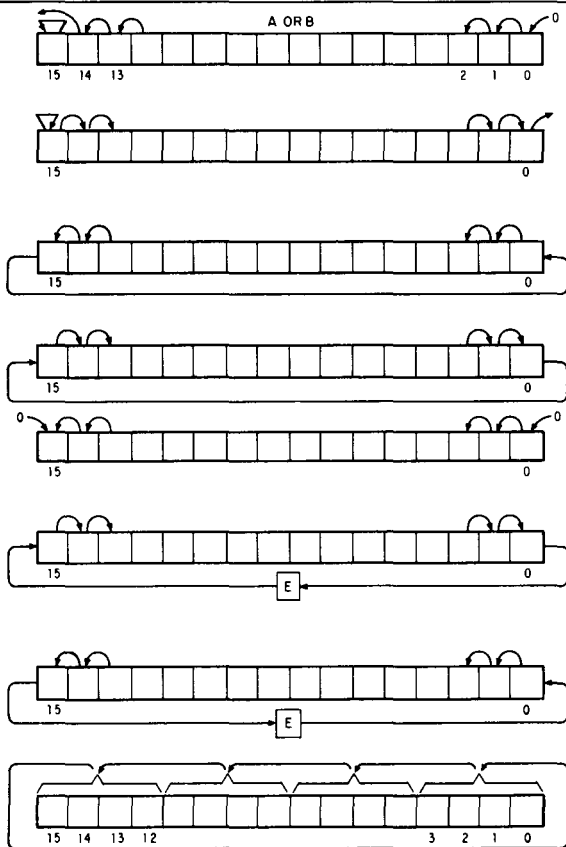
CMA/B. To complement A or B, the register is read onto the R Bus, the complement function (CMF) reverses each bit before being released to the T Bus, and the T Bus is stored back into the A or B Register.

Table 3.1.

INSTRUCTION	FUNCTIONS	
A/BLS	$\overline{\text{SLM}} \bullet \text{RB}(0-13)$ $\text{RB15} - \text{TB15}$	
A/BRS	$\overline{\text{SRM}} \bullet \text{RB}(1-15)$ $\text{RB15} - \text{TB15}$	
RA/BL	$\overline{\text{SLM}} \bullet \text{RB}(0-13)$ $\text{SL14} \bullet \text{RB14}$ $\text{RLL} \bullet \text{RB15}$	
RA/BR	$\overline{\text{SRM}} \bullet \text{RB}(1-15)$ $\text{RRS} \bullet \text{RB0}$	
A/BLR	$\overline{\text{SLM}} \bullet \text{RB}(0-13)$	
ERA/B	$\overline{\text{SRM}} \bullet \text{RB}(1-15)$ $\text{E} - \text{TB15}$ $\text{RB0} - \text{E}$	
ELA/B	$\overline{\text{SLM}} \bullet \text{RB}(0-13)$ $\text{SL14} \bullet \text{RB14}$ $\text{E} - \text{TB0}$ $\text{RB15} - \text{E}$	
A/BLF	$\overline{\text{RL4}} \bullet \text{RB}(0-15)$	
SLM	Shift Left Magnitude	RB R Bus
SRM	Shift Right Magnitude	TB T Bus
RLL	Rotate Left to Least significant bit	SL Shift Left
RRS	Rotate Right to Sign bit	RL Rotate Left

Shift-Rotate Functions

DIAGRAMS



CCA/B. The procedures of the two preceding paragraphs are combined to clear and complement an accumulator; i.e., with no Read, R and S Buses remain all-zero, and the complement function reverses this state to all ones on the T Bus. The T Bus is then stored into the A or B Register.

SEZ. If Bit 5 of the T-Register is a one, the Extend flip-flop and Bit 0 of the T-Register (Reverse Skip Sense) are looked at by the computer logic, causing the Carry flip-flop to be set if: a) both bits are zero, b) both bits are one. Although the next three instructions described below can alter the state of the Extend flip-flop, the test is completed before the alteration.

CLE. If Bits 6 and 7 of the T-Register encode the Clear E instruction, a reset signal is generated during the latter half of T3 to reset the Extend flip-flop.

CME. If Bits 6 and 7 of the T-Register encode Complement E, the state of the Extend flip-flop is reversed during the latter half of T3.

CCE. If Bits 6 and 7 of the T-Register encode Clear and Complement E, the Extend flip-flop is set during the latter half of T3.

SSA/B. If Bit 4 of the T-Register is a one, the A or B Register is read onto the R Bus. Bit 15 of the R Bus (sign bit) and Bit 0 of the T-Register (Reverse Skip Sense) are tested. The Carry flip-flop will be set if both bits are zero (meaning: "skip if sign bit is zero"), or if both bits are one (meaning: "skip if sign bit is not zero"). This is accomplished during T4.

SLA/B. If Bit 3 of the T-Register is a one, the A or B Register is read onto the R Bus. Bit 0 of the R Bus (least significant bit) and Bit 0 of the T-Register (Reverse Skip Sense) are tested. The Carry flip-flop will be set if both bits are zero (meaning: "skip if least significant bit is zero"), or if both bits are one (meaning: "skip if least significant bit is not zero"). This is accomplished during T4. The combination of SLA/B, SZA/B, and RSS is a special case.

INA/B. If Bit 2 of the T-Register is a one, the A or B Register is read onto the R Bus, and a "one" is read onto the S Bus. These are combined by an add function (ADF) and stored back into the A or B Register during the latter half of T5.

SZA/B. If Bit 1 of the T-Register is a one, the A or B Register is read onto the R Bus and transmitted to the T Bus. All bits of the T Bus are applied to an "inclusive or" gate. The output of this gate and Bit 0 of the T-Register are tested. The Carry flip-flop will be set if both TR0 and the gate output are zero (meaning: "skip if ac-

cumulator is zero'), or if both TR0 and the gate output are one (meaning: "skip if accumulator is not zero").

3.2.5 INPUT/OUTPUT INSTRUCTIONS

Like the Register Reference instructions, Input/Output instructions, as shown by Figure 3.14, are fully executed in only one phase (Fetch). The Interrupt phase, shown at the bottom of Figure 3.14, is not involved in the execution of these instructions. It is separately discussed at the end of this chapter.

The following descriptions will concentrate on actions occurring during Time Periods T3, T4, and T5, since as can be seen from Figure 3.14, the actions during other Time Periods are nearly identical from instruction to instruction. That is, the T-Register is cleared during T0, the Instruction Register is cleared during T1, and the P and M Registers are incremented by one (or two, if a Carry bit is present) during T6 and T7. The method of incrementing by one and by two was described earlier. In all cases, Bits 10 through 15 of the T-Register are transferred to the Instruction Register during T2.

HLT. If Bits 8, 7, 6 of the T-Register encode the Halt instruction, these bits cause the Run flip-flop to be reset during the latter half of T7.

STF. During T3 a Set Flag signal is routed to all input/output interface cards, and will set the Flag flip-flop of the card which is currently enabled by the Select Code (Bits 0 through 5 of the T-Register).

CLF. During T4 a Clear Flag signal is routed to all input/output interface cards, and will reset the Flag flip-flop of the card which is currently enabled by the Select Code.

SFC. A Skip if Flag Clear signal (SFC) is routed to the selected interface card beginning at T3. The interface card will return a Skip Flag signal (SKF) during T4 if its Flag flip-flop is not set. This signal sets the Carry flip-flop to cause a skip during T6 and T7.

SFS. A Skip if Flag Set signal (SFS) is routed to the selected interface card beginning at T3. The interface card will return a Skip Flag signal (SKF) during T4 if its Flag flip-flop is set. This signal sets the Carry flip-flop to cause a skip during T6 and T7.

MIA/B. During T4 and T5 an IOIC signal (I/O Input Control) transfers the input data from the interface Buffer register to the S Bus.

During the same time the A or B Register is read onto the R Bus, and the R and S Bus data is combined by the "inclusive or" function (IOF) and applied to the T Bus. The result (a "merge", or "inclusive or") is stored back into the A or B Register. If Bit 9 of the T-Register is a one, a Clear Flag signal (CLF) is routed to the Flag flip-flop of the selected interface card.

LIA/B. The action for LIA/B (Load Input into A or B) is the same as described for MIA/B in the preceding paragraph, except that nothing is read onto the R Bus. The "inclusive or" function therefore transmits the R Bus unchanged to the T Bus for storing into the A or B Register. As for MIA/B, Bit 9 can clear the Flag flip-flop.

OTA/B. During T4 and T5 the A or B Register is read onto the R Bus, which in turn is transferred by an IOOC signal (I/O Output Control) to the interface Buffer register. As for MIA/B, Bit 9 can clear the Flag flip-flop.

STC. A Set Control signal is routed to all input/output interface cards, and during T4 will set the Control flip-flop of the interface card which is currently enabled by the Select Code (Bits 0 through 5 of the T-Register).

CLC. A Clear Control signal is routed to all interface cards during T4, and will reset the Control flip-flop of the interface card currently enabled by the Select Code.

STO. A Set Flag signal during T3, combined with the Select Code for the Overflow flip-flop (01_8), sets the Overflow flip-flop.

CLO. A Clear Flag signal during T4, combined with the Select Code for the Overflow flip-flop (01_8), resets the Overflow flip-flop.

SOC. During T3, a Skip if Flag Clear signal (SFC), combined with the Select Code for Overflow, tests the state of the Overflow flip-flop. If this flip-flop is in the reset state, a Skip Flag signal (SKF) sets the Carry flip-flop at T4, to cause a skip at T6 and T7.

SOS. During T3, a Skip if Flag Set signal (SFS), combined with the Select Code for Overflow, tests the state of the Overflow flip-flop. If this flip-flop is in the set state, a Skip Flag signal (SKF) sets the Carry flip-flop at T4, to cause a skip at T6 and T7.

3.2.6 INTERRUPT PHASE

The actions occurring during the Interrupt phase (Phase 4) are shown at the bottom of Figure 3.14. Two operations are accomplished during the Interrupt phase:

a. The P-Register is decremented. This is done so that any instruction which has not been fully executed at the time of interrupt will be repeated. On the other hand, if the instruction is fully executed (which means that the P-Register has been advanced for the next instruction), it is still necessary to decrement. This is because the P-Register is incremented for a second time following execution of the instruction contained in the interrupt location.

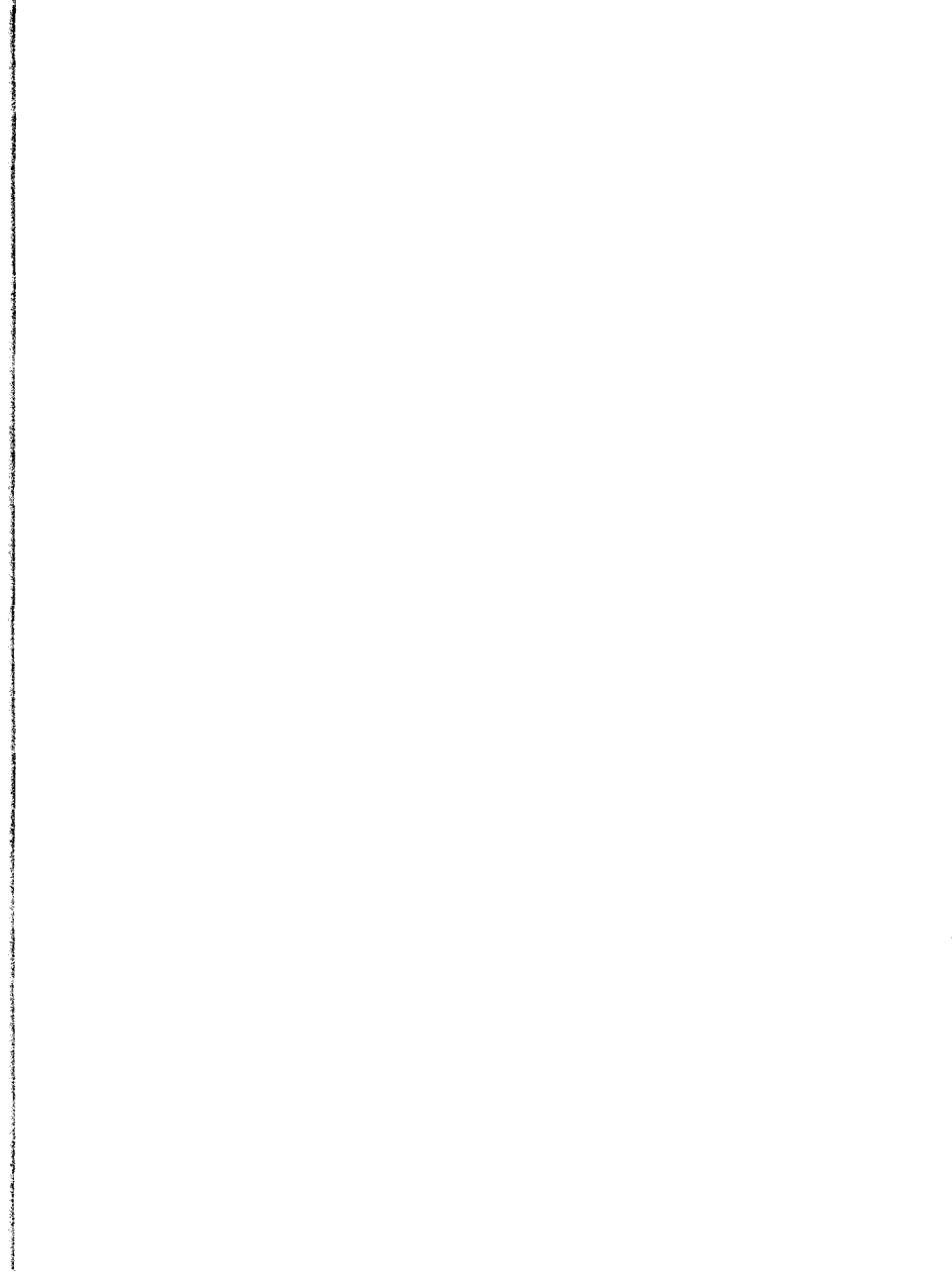
b. The "interrupt address" must be transferred into the M-Register, and Phase 1 is set. This causes the instruction contained in the interrupt location to be read out of memory for execution during the next machine phase. Note that the interrupt address is not placed into the P-Register. While the instruction in the interrupt location is being executed, the P-Register remains at the value one lower than the point at which interrupt occurred.

Decrementing the P-Register is accomplished by complementing, incrementing, then complementing again. In simplified form, using only four binary digits for an example, this process is:

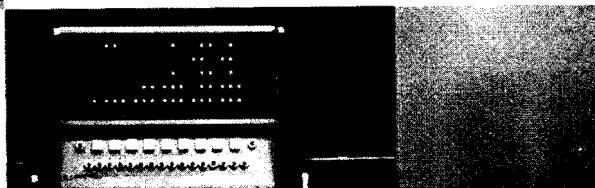
Original Value:	0110	(6 ₈)
Complement:	1001	
Increment:	1010	
Complement:	0101	(5 ₈)

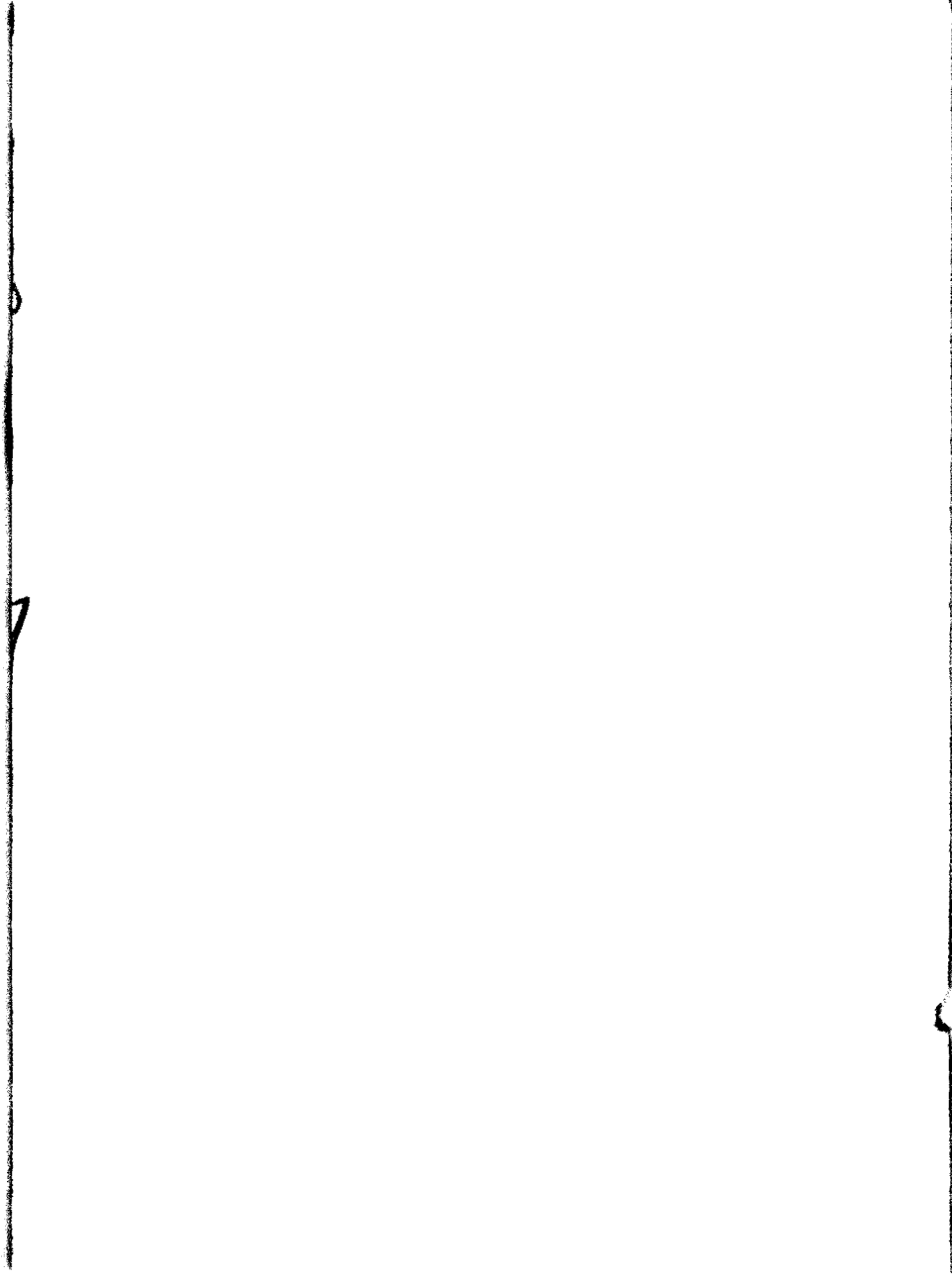
During T1 and T2 of the Interrupt phase (remember that there is no Read/Write memory cycle), the P-Register is read onto the R Bus. The complement function (CMF) reverses all bits before application to the T Bus, and then the T Bus is stored back into the P-Register. During T3 and T4 the P-Register is again read onto the R Bus. A "one" read onto the S Bus is combined with this by the add function (ADF), and the incremented result is stored back into the P-Register. During T5, the P-Register is read onto the R Bus for the third time, is complemented, applied to the T Bus, and stored back into the P-Register.

The interrupt address is placed into the M-Register during T7. Since no interrupt address is greater than 77₈, M-Register Bits 6 through 15 are first reset. The interrupt address is read directly onto the T Bus from Input/Output Control logic (see Figure 3.1), and Bits 0 through 5 are stored into the M-Register. Setting the Phase 1 condition completes the Interrupt phase.



Assembler Reference Manual





CONTENTS

INTRODUCTION	V
CHAPTER 1 GENERAL DESCRIPTION	1-1
1.1 Assembly Processing	1-1
1.2 Symbolic Addressing	1-1
1.3 Program Relocation	1-3
1.4 Program Location Counters	1-3
1.5 Assembly Options	1-4
CHAPTER 2 INSTRUCTION FORMAT	2-1
2.1 Statement Characteristics	2-1
2.2 Label Field	2-3
2.3 Opcode Field	2-5
2.4 Operand Field	2-5
2.5 Comments Field	2-13
CHAPTER 3 MACHINE INSTRUCTIONS	3-1
3.1 Memory Reference	3-1
3.2 Register Reference	3-4
3.3 Input/Output, Overflow, and Halt	3-7
3.4 Extended Arithmetic Unit	3-11
CHAPTER 4 PSEUDO INSTRUCTIONS	4-1
4.1 Assembler Control	4-1
4.2 Object Program Linkage	4-8
4.3 Address and Symbol Definition	4-11
4.4 Constant Definition	4-17
4.5 Storage Allocation	4-21
4.6 Assembly Listing Control	4-21
4.7 Arithmetic Subroutine Calls	4-24
CHAPTER 5 ASSEMBLER INPUT AND OUTPUT	5-1
5.1 Control Statement	5-1
5.2 Source Program	5-2
5.3 Binary Output	5-2
5.4 List Output	5-2
5.5 Operating Instructions	5-3
5.6 Object Program Loading	5-8
5.7 Error Messages	5-9

APPENDIX	A	HP Character Set	A-1
APPENDIX	B	Summary of Instructions	B-1
APPENDIX	C	Alphabetical List of Instructions	C-1
APPENDIX	D	Sample Problems	D-1
APPENDIX	E	System Input/Output Subroutines	E-1
APPENDIX	F	Formatter	F-1
APPENDIX	G	Cross Reference Table Generator	G-1
APPENDIX	H	Consolidated Coding Sheet	H-1

INTRODUCTION

The Assembler and the Extended Assembler translate symbolic source language instructions into an object program for execution on the computer. The source language provides mnemonic machine operation codes, assembler directing pseudo codes, and symbolic addressing. The assembled program may be absolute or relocatable.

The source program may be assembled as a complete entity or it may be subdivided into several subprograms (or a main program and several subroutines), each of which may be assembled separately. The loader of the Basic Control System loads the program and links the subprograms as required. The Basic Binary Loader loads programs in absolute form.

Input for the Assembler is prepared on paper tape; the Assembler punches the binary program on paper tape in a format acceptable to the loader.

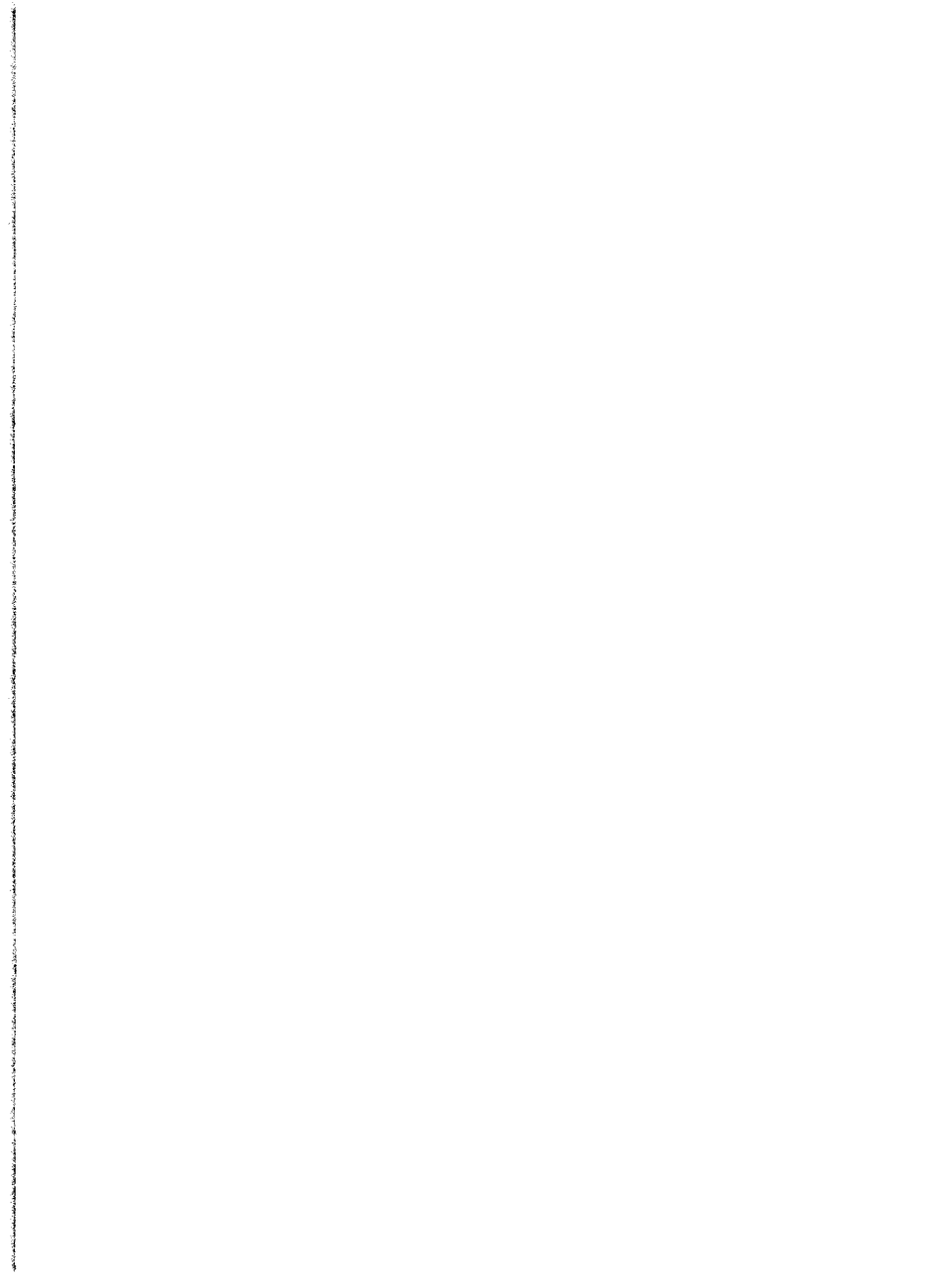
The minimum equipment configuration required to use the Assembler is as follows:

2116B, 2115A, or 2114A Computer with 4K memory
2752A Teleprinter

The minimum configuration for the Extended Assembler is:

2116B, 2115A, or 2114A Computer with 8K memory
2752A Teleprinter

Both the Assembler and Extended Assembler are available for configurations which include the Magnetic Tape Unit.



1.1 ASSEMBLY PROCESSING

The Assembler is a two pass system, or, if both punch and list output are requested, a three pass system on a minimum configuration. A pass is defined as a processing cycle of the source program input.

In the first pass, the Assembler creates a symbol table from the names used in the source statements. It also checks for certain possible error conditions and generates diagnostic messages if necessary.

During pass two, the Assembler again examines each statement in the source program along with the symbol table and produces the binary program and a program listing. Additional diagnostic messages may also be produced.

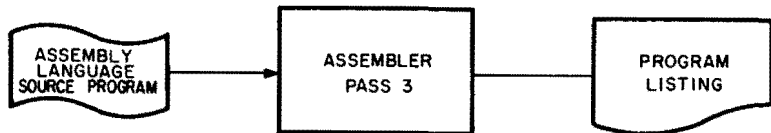
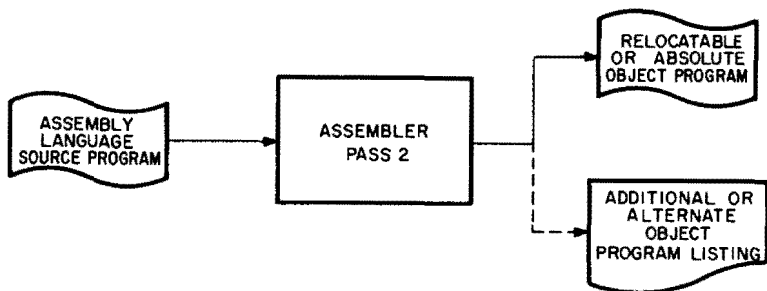
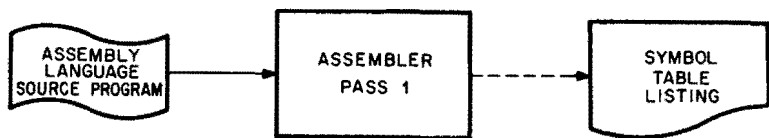
If only one output device is available and if both the binary output and the list output are requested, the listing function is deferred and performed as pass three.

When using the Assembler with a magnetic tape the source program is written on the tape during the first pass; the tape is backspaced and the second pass executed.

1.2 SYMBOLIC ADDRESSING

Symbols may be used for referring to machine instructions, data, constants, and certain other pseudo operations. A symbol represents the address for a computer word in memory. A symbol is defined when it is used as a label for a location in the program, a name of a common storage segment, the label of a data storage area or constant, the label of an absolute or relocatable value, or a location external to the program.

Through use of simple arithmetic operators, symbols may be combined with other symbols or numbers to form an expression which may identify a location other than that specifically named by a symbol. Symbols appearing in operand expressions, but not specifically defined, and symbols that are defined more than once are considered to be in error by the Assembler.



HP ASSEMBLER PROCESSING

1.3 PROGRAM RELOCATION

Programs may be relocated in core by the Basic Control System loader; the location of the program origin and all subsequent instructions is determined at the time the program is loaded.

A relocatable program is assembled assuming a starting location of zero. All other instructions and data areas are assembled relative to this zero base. When the program is loaded, the relocatable operands are adjusted to correspond with the actual locations assigned by the loader.

The starting locations of the common storage area and the base page portion of the program are always established by the loader. References to the common area are common relocatable. References to the base page portion of the program are base page relocatable. If a program refers to the common area or makes use of the base page via the ORB pseudo instruction, the program must also be relocatable.

If a program is to be relocatable, all subprograms comprising the program must be relocatable; all memory reference operands must be relocatable expressions or literals, or have an absolute value of less than 100₈.

1.4 PROGRAM LOCATION COUNTERS

The Assembler maintains a counter, called the program location counter, that it uses to assign consecutive memory addresses to source statements.

The initial value of the program location counter is established according to the use of either the NAM or ORG pseudo operation at the start of the program. The NAM operation causes the program location counter to be set to zero for a relocatable program; the ORG operation specifies the absolute starting location for an absolute program.

Through use of the ORB pseudo operation a relocatable program may specify that certain operations or data areas be allocated to the base page. If so, a separate counter, called the base page location counter, is used in assigning these locations.

1.5 ASSEMBLY OPTIONS

Parameters specified with the first statement, the control statement, define the output to be produced by the Assembler:†

Absolute – The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The program is a complete entity; external symbols, common storage references, and entry points are not permitted.

Relocatable – The program may be located anywhere in memory. All operands which refer to memory locations are adjusted as the program is loaded. Operands, other than those referring to the first 64 locations, must be relocatable expressions. Subprograms may contain external symbols and entry points, and may refer to common storage.

Binary output – An absolute or relocatable program is to be punched on paper tape.

List output – A program listing is produced either during pass two or pass three.

Table print – List the symbol table at the end of the first pass.

Selective assembly – Sections of the program may be included or excluded at assembly time depending on the option used.

† See Chapter 5 for complete details.

A source language statement consists of a label, an operation code, an operand, and comments. The label is used when needed as a reference by other statements. The operation code may be a mnemonic machine operation or an assembly directing pseudo code. An operand may be an expression consisting of an alphanumeric symbol, a number, a special character, or any of these combined by arithmetic operations. (For the Extended Assembler, an operand may also be a literal.) Indicators may be appended to the operand to specify certain functions such as indirect addressing. The comments portion of the statement is optional.

2.1 STATEMENT CHARACTERISTICS

The fields of the source statement appear in the following order:

- Label
- Opcode
- Operand
- Comments

Field Delimiters

One or more spaces separate the fields of a statement. An end-of-statement mark terminates the entire statement. On paper tape this mark is a return, (CR), and line feed, (LF).† A single space following the end-of-statement mark from the previous source statement is the null field indicator of the label field.

Character Set

The characters that may appear in a statement are as follows:

- A through Z
- 0 through 9
- . (period)
- * (asterisk)

† A circled symbol (e.g., (CR)) represents an ASCII code or Teleprinter key.

PROGRAMME		DATE	PROGRAM	PAGE	OF

[illegible]

SAMPLE CODING FORM
(Actual Size 11 × 13-1/2)

- + (plus)
- (minus)
- , (comma)
- = (equals)
- () (parentheses)
- (space)

Any other ASCII characters may appear in the Remarks field (See Appendix A).

The letters A through Z, the numbers 0 through 9, and the period may be used in an alphanumeric symbol. In the first position in the Label field, an asterisk indicates a comment; in the Operand field, it represents the value of the program location counter for the current instruction. The plus and minus are used as operators in arithmetic address expressions. The comma separates several operation codes, or an expression and an indicator in the Operand field. An equals sign indicates a literal value. The parentheses are used only in the COM pseudo instruction.

Spaces separate fields of a statement. They may also be used to establish the format of the output list. Within a field they may be used freely when following +, -, ,, or (.

Statement Length

A statement may contain up to 80 characters including blanks, but excluding the end-of-statement mark. Fields beginning in characters 73 - 80 are not processed by the Assembler.

2.2 LABEL FIELD

The Label field identifies the statement and may be used as a reference point by other statements in the program.

The field starts in position one of the statement; the first position following an end-of-statement mark for the preceding statement. It is terminated by a space. A space in position one is the null field indicator for the label field; the statement is unlabeled.

Label Symbol

A label must be symbolic. It may have one to five characters consisting of A through Z, 0 through 9, and the period. The

first character must be alphabetic or a period. A label of more than five characters could be entered on the source language tape, but the Assembler flags this condition as an error and truncates the label from the right to five characters.

Examples:

Label	Operation	Operand	Comments
1	5	15	25 30 35 40 45 50
AT	LDA		NO LABEL
.ABCD			VALID LABEL
.1234			VALID LABEL
A.123			VALID LABEL
..			VALID LABEL
1.AB			ILLEGAL LABEL - FIRST CHARACTER
			NUMERIC.
ABC123			ILLEGAL LABEL - TRUNCATED TO
			ABC12.
A*BC			ILLEGAL LABEL - ASTERISK NOT
			ALLOWED IN LABEL.
^ABCT			NO LABEL - THE ASSEMBLER ATTEMPTS
			TO INTERPRET ABC AS AN OPERATION
			CODE.

Each label must be unique within the program; two or more statements may not have the same symbolic name. Names which appear in the Operand field of an EXT or COM pseudo instruction may not also be used as statement labels in the same subprogram.

Examples:

Label	Operation	Operand	Comments
1	5	15	25 30 35 40 45 50
	COM	ACOM(20),BC(30)	
LB	EQU	160	VALID LABEL
	EXT	XL1,XL2	
START	LDA	LB	VALID LABEL
N25			VALID LABEL
XL2			ILLEGAL LABEL - USED IN EXT.
BC			ILLEGAL LABEL - USED IN COM.
N25			ILLEGAL LABEL - PREVIOUSLY
			DEFINED.

† The caret symbol, ^, indicates the presence of a space.

Asterisk

An asterisk in position one indicates that the entire statement is a comment. Positions 2 through 80 are available; however, positions 1 through 68 only are printed as part of the assembly listing on the 2752A Teleprinter. An asterisk within the Label field is illegal in any position other than one.

2.3 OPCODE FIELD

The operation code defines an operation to be performed by the computer or the Assembler. The Opcode field follows the Label field and is separated from it by at least one space. If there is no label, the operation code may begin anywhere after position one. The Opcode field is terminated by a space immediately following an operation code. Operation codes are organized in the following categories:

Machine operation codes

- Memory Reference

- Register Reference

- Input/Output, Overflow, and Halt

- Extended Arithmetic Unit

Pseudo operation codes

- Assembler control

- Object program linkage

- Address and symbol definition

- Constant definition

- Storage allocation

- Arithmetic subroutine calls

- Assembly Listing Control (Extended Assembler)

Operation codes are discussed in detail in Chapters 3 and 4.

2.4 OPERAND FIELD

The meaning and format of the Operand field depend on the type of operation code used in the source statement. The field follows the Opcode field and is separated from it by at least one space. It is terminated by a space except when the space follows , + - (or, if there are no comments, by an end-of-statement mark.

The Operand field may contain an expression consisting of one of the following:

Single symbolic term

Single numeric term

Asterisk

Combination of symbolic terms, numeric terms, and the asterisk jointed by the arithmetic operators + and -.

An expression may be followed by a comma and an indicator.

Programs being assembled by the Extended Assembler may also contain a literal value in the Operand field.

Symbolic Terms

A symbolic term may be one to five characters consisting of A through Z, 0 through 9, and the period. The first character must be alphabetic or a period.

Examples:

Label	Operation	Operand	Comments
	LDA	A1234	VALID IF DEFINED
	ADA	B.1	VALID IF DEFINED
	JMP	ENTRY	VALID IF DEFINED
	STA	1ABC	ILLEGAL OPERAND FIRST CHARACTER
			NUMERIC.
	STB	ABCDEF	ILLEGAL OPERAND MORE THAN FIVE
			CHARACTERS.

A symbol used in the Operand field must be a symbol that is defined elsewhere in the program in one of the following ways:

As a label in the Label field of a machine operation

As a label in the Label field of a BSS, ASC, DEC, OCT, DEF, ABS, EQU or REF pseudo operation

As a name in the Operand field of a COM or EXT pseudo operation

As a label in the Label field of an arithmetic subroutine pseudo operation

The value of a symbol is absolute or relocatable depending on the assembly option selected by the user. The Assembler assigns a value to a symbol as it appears in one of the above fields of a statement. If a program is to be loaded in absolute form, the values assigned by the assembler remain fixed. If the program is to be relocated, the actual value of a symbol is established on loading. A symbol may also be made absolute through use of the EQU pseudo instruction.

A symbolic term may be preceded by a plus or minus sign. If preceded by a plus or no sign, the symbol refers to its associated value. If preceded by a minus sign, the symbol refers to the two's complement of its associated value. A single negative symbolic operand may be used only with the ABS pseudo operation.

Numeric Terms

A numeric term may be decimal or octal. A decimal number is represented by one to five digits within the range 0 to 32767. An octal number is represented by one to six octal digits followed by the letter B; (0 to 177777B).

If a numeric term is preceded by a plus or no sign, the binary equivalent of the number is used in the object code. If preceded by a minus sign, the two's complement of the binary equivalent is used. A negative operand may be used only with the DEC, OCT, and ABS pseudo operations.

In an absolute program, the maximum value of a numeric operand depends on the type of machine or pseudo instruction. In a relocatable program, the value of a numeric operand may not exceed 77B. Numeric operands are absolute. Their value is not altered by the assembler or the loader.

Asterisk

An asterisk in the Operand field refers to the value in the program location counter (or base page location counter) at the time the source program statement is encountered. The asterisk is considered a relocatable term in a relocatable program.

Expression Operators

The asterisk, symbols, and numbers may be joined by the arithmetic operators + and - to form arithmetic address expressions. The Assembler evaluates an expression and produces an absolute or relocatable value in the object code.

Examples:

Label	Operation	Operand	Comment
	LDA	SYM+6	ADD 6 TO THE VALUE OF SYM
	ADA	SYM-3	SUBTRACT 3 FROM THE VALUE OF SYM
	*		
	JMP	*+5	ADD 5 TO THE CONTENTS OF THE PROGRAM LOCATION COUNTER.
	*		
	STB	-A+C-4	ADD - VALUE OF A, THE VALUE OF C AND SUBTRACT 4.
	*		
	STA	XTA-*	SUBTRACT VALUE OF PROGRAM LOCATION COUNTER FROM VALUE OF XTA.

Evaluation of Expressions

An expression consisting of a single operand has the value of that operand. An expression consisting of more than one operand is reduced to a single value. In expressions containing more than one operator, evaluation of the expression proceeds from left to right. The algebraic expression $A-(B-C+5)$ must be represented in the Operand field as $A-B+C-5$. Parentheses are not permitted in operand expressions for the grouping of operands.

The range of values that may result from an operand expression depends on the type of operation. The Assembler evaluates expressions as follows:†

Pseudo Operations	modulo $2^{15}-1$
Memory Reference	modulo $2^{10}-1$
Input/Output	$2^6 - 1$ (maximum value)

† The evaluation of expressions by the Assembler is compatible with the addressing capability of the hardware instructions (e.g., up to 32K words through Indirect Addressing). The user must take care not to create addresses which exceed the memory size of the particular configuration.

Expression Terms

The terms of an expression are the numbers and the symbols appearing in it. Decimal and octal integers, and symbols defined as being absolute in an EQU pseudo operation are absolute terms. The asterisk and all symbols that are defined in the program are relocatable or absolute depending on the type of assembly. Symbols that are defined as external may appear only as single term expressions.

Within a relocatable program, terms may be program relocatable, base page relocatable, or common relocatable. A symbol that names an area of common storage is a common relocatable term. A symbol that is allocated to the base page is a base page relocatable term. A symbol that is defined in any other statement is a program relocatable term. Within one expression all relocatable terms must be base page relocatable, program relocatable, or common relocatable; the three types may not be mixed.

Absolute and Relocatable Expressions

An expression is absolute if its value is unaffected by program relocation. An expression is relocatable if its value changes according to the location into which the program is loaded. In an absolute program, all expressions are absolute. In a relocatable program, an expression may be base page relocatable, program relocatable, common relocatable, or absolute (if less than 100g) depending on the definition of the terms composing it.

Absolute Expressions

An absolute expression may be any arithmetic combination of absolute terms. It may also contain relocatable terms alone, or in combination with absolute terms. If relocatable terms do appear, there must be an even number of them; they must be of the same type; and they must be paired by sign (a negative term for each positive term). The paired terms do not have to be contiguous in the expression. The pairing of terms by type cancels the effect of relocation; the value represented by the pair remains constant.

An absolute expression reduces to a single absolute value. The value of an absolute multiterm expression may be negative only for ABS pseudo operations. A single numeric term also may be negative in an OCT or DEC pseudo instruction. In a relocatable program the value of an absolute expression must be less than 100g for instructions that reference memory locations (Memory Reference, DEF, Arithmetic subroutine calls).

Examples:

If P_1 and P_2 are program relocatable terms; B_1 and B_2 , base page relocatable; C_1 and C_2 , common relocatable; and A , an absolute term; then the following are absolute terms:

$A - C_1 + C_2$	$A - P_1 + P_2$	$C_1 - C_2 + A$
$A + A$	$P_1 - P_2$	$B_1 - B_2$
$* - P_1$	$B_1 - B_2 - A$	$-C_1 + C_2 + A$
$B_1 - *$	$-P_1 + P_2$	$-A - P_1 + P_2$

The asterisk is base page relocatable or program relocatable depending on the location of the instruction.

Relocatable Expressions

A relocatable expression is one whose value is changed by the loader. All relocatable expressions must have a positive value.

A relocatable expression may contain any odd number of relocatable terms, alone, or in combination with absolute terms. All relocatable terms must be of the same type. Terms must be paired by sign with the odd term being positive.

A relocatable expression reduces to a single positive relocatable term, adjusted by the values represented by the absolute terms and paired relocatable terms associated with it.

Examples:

If P_1 , P_2 , and P_3 are program relocatable terms; B_1 , B_2 , and B_3 , base page relocatable; C_1 , C_2 and C_3 , common relocatable; and A , an absolute term; then the following are relocatable terms:

$P_1 - A$	$C_1 - A$	$B_1 + A$
$P_1 - P_2 + P_3$	$C_1 - C_2 + C_3$	$C_1 + A$
$* + A$	$* - P_1 + P_2$	$* - A$
$A + B_1$	$A + C_1$	$-A - P_1 + P_2 + P_3$
$B_1 - B_2 + B_3 - A$	$C_1 - C_2 + C_3 - A$	$A + *$
$* + P_1 - *$	$P_1 - P_2 + *$	$-C_1 + C_2 + C_3$

Literals

Actual literal values may be specified as operands in relocatable programs to be assembled by the Extended Assembler. The Extended Assembler converts the literal to its binary value, assigns an address to it, and substitutes this address as the operand. Locations assigned to literals are those immediately following the last location used by the program.

A literal is specified by using an equal sign and a one-character identifier defining the type of literal. The actual literal value is specified immediately following this identifier; no spaces may intervene.

The identifiers are:

- =D a decimal integer, in the range -32767 to 32767, including zero.†
- =F a floating point number; any positive or negative real number in the range 10^{-38} to 10^{38} , including zero.†
- =B an octal integer, one to six digits, $b_1b_2b_3b_4b_5b_6$, where b_1 may be 0 or 1, and b_2-b_7 may be 0 to 7.†
- =A two ASCII characters.†
- =L an expression which, when evaluated, will result in an absolute value. All symbols appearing in the expression must be previously defined.

If the same literal is used in more than one instruction, only one value is generated, and all instructions using this literal refer to the same location.

Literals may be specified only in the following memory reference instructions and pseudo instructions:

ADA	ADB	AND	MPY	} may use =D, =B, =A, =L
LDA	LDB	XOR	DIV	
CPA	CPB	IOR		
DLD	FAD	} may use =F		
FMP	FSB			
FDV				

† See CONSTANT DEFINITION, Section 4.4.

LDA	=D7980	A-Register is loaded with the binary equivalent of 7980 ₁₀ .
IOR	=B777	Inclusive or is performed with contents of A-Register and 777 ₈ .
LDA	=ANO	A-Register is loaded with binary representation of ASCII characters NO.
LDB	=LZETZ-ZOOM+68	B-Register is loaded with the value resulting from the absolute expression.
FMP	=F39.75	Contents of A- and B-Registers multiplied by floating point constant 39.75.

The HP computers provide an indirect addressing capability for Memory Reference instructions. The operand portion of an indirect instruction contains an address of another location rather than an actual operand. The secondary location may be the operand or it may be indirect also and give yet another location, and so forth. The chaining ceases when a location is encountered that does not contain an indirect address. Indirect addressing provides a simplified method of address modifications as well as allowing access to any location in core.

The Assembler allows specification of indirect addressing by appending a comma and the letter I to any Memory Reference operand other than one referring to an external symbol. The actual operand of the instruction may be given in a DEF pseudo operation; this pseudo operation may also be used to indicate further levels of indirect addressing.

Label	Operand	Comment
AB	LDA SAM, I	EACH TIME THE ISZ IS EXECUTED,
AC	ADA SAM, I	THE EFFECTIVE OPERAND OF AB AND
AD	ISZ SAM	AC CHANGE ACCORDINGLY.
	*	
	*	
SAM	DEF ROGER	

ceed 52 characters, the width of the source language portion of the listing. Statements consisting solely of comments may contain up to 68 characters including the asterisk in the first position. On the list output, statements consisting entirely of comments begin in position 5 rather than 21 as with other source statements.

If there is no operand present the Comments field should be omitted in the NAM and END pseudo operations and in the input/output statements, SOC, SOS, and HLT. If a comment is used, the Assembler attempts to interpret it as an operand.

The HP Assembler language machine instruction codes take the form of three-letter mnemonics. Each source statement corresponds to a machine operation in the object program produced by the Assembler.

Notation used in representing source language instruction is as follows:

label	Optional statement label
m	Memory location -- an expression
I	Indirect addressing indicator
sc	Select code -- an expression
C	Clear interrupt flag indicator
comments	Optional comments
[]	Brackets defining a field or portion of a field that is optional
{ }	Brackets indicating that one of the set may be selected.
lit	literal

3.1 MEMORY REFERENCE

Memory Reference instructions perform arithmetic, logical and jump operations on the contents of the locations in core and the registers. An instruction may directly address the 2048 words of the current and base pages. If required, indirect addressing may be utilized to refer to all 32,768 words of memory. Expressions in the operand field are evaluated modulo 2^{10} .

If the program is to be assembled in relocatable form, the operand field may contain relocatable expressions or absolute expressions which are less than 100g in value. If the program is to be absolute, the operands may be any expressions consistent with the location of the program. Literals may not be used in an absolute program. Absolute programs must be complete entities; they may not refer to external subroutines or common storage.

Jump and Increment-Skip

Jump and Increment-Skip instructions may alter the normal sequence of program execution.

label	JMP	m [, I]	comments
-------	-----	---------	----------

Jump to m. Jump indirect inhibits interrupt until the transfer of control is complete.

label	JSB	m [, I]	comments
-------	-----	---------	----------

Jump to subroutine. The address for label+1 is placed into the location represented by m and control transfers to m+1. On completion of the subroutine, control may be returned to the normal sequence by performing a JMP m, I.

label	ISZ	m [, I]	comments
-------	-----	---------	----------

Increment, then skip if zero. ISZ adds 1 to the contents of m. If m then equals zero, the next instruction in memory is bypassed.

Add, Load, and Store

Add, Load, and Store instructions transmit and alter the contents of memory and of the A- and B-Registers. A literal, indicated by 'lit', may be either =D, =B, =A, or =I type.

label	ADA	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

Add the contents of m to A.

label	ADB	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

Add the contents of m to B.

label	LDA	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

Load A from m.

label	LDB	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

Load B from m.

label	STA	m [, I]	comments
-------	-----	---------	----------

Store contents of A in m.

label	STB	m [, I]	comments
-------	-----	---------	----------

Store contents of B in m.

In each instruction, the contents of the sending location is unchanged after execution.

Logical Operations

The Logical instructions allow bit manipulation and the comparison of two computer words.

label	AND	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

The logical product of the contents of m and the contents of A are placed in A.

label	XOR	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

The modulo-two sum (exclusive "or") of the bits in m and the bits in A is placed in A.

label	IOR	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

The logical sum (inclusive "or") of the bits in m and the bits in A is placed in A.

label	CPA	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

Compare the contents of m with the contents of A. If they differ, skip the next instruction; otherwise, continue.

label	CPB	$\left\{ \begin{array}{l} m [, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	---	----------

Compare the contents of m with the contents of B. If they differ, skip the next instruction; otherwise, continue.

3.2 REGISTER REFERENCE

The Register Reference instructions include a Shift-Rotate group, an Alter-Skip group, and NOP (no-operation). With the exception of NOP, they have the capability of causing several actions to take place during one memory cycle. Multiple operations within a statement are separated by a comma.

Shift-Rotate Group

This group contains 19 basic instructions that can be combined to produce more than 500 different single cycle operations.

CLE	Clear E to zero
ALS	Shift A left one bit, zero to least significant bit. Sign unaltered
BLS	Shift B left one bit, zero to least significant bit. Sign unaltered
ARS	Shift A right one bit, extend sign; sign unaltered.
BRS	Shift B right one bit, extend sign; sign unaltered.
RAL	Rotate A left one bit
RBL	Rotate B left one bit
RAR	Rotate A right one bit
RBR	Rotate B right one bit
ALR	Shift A left one bit, clear sign, zero to least significant bit
BLR	Shift B left one bit, clear sign, zero to least significant bit
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	Skip next instruction if least significant bit in A is zero
SLB	Skip next instruction if least significant bit in B is zero

These instructions may be combined as follows:

label	$\left\{ \begin{array}{l} \text{ALS} \\ \text{ARS} \\ \text{RAL} \\ \text{RAR} \\ \text{ALR} \\ \text{ALF} \\ \text{ERA} \\ \text{ELA} \end{array} \right\}$	[CLE]	[SLA]	,	$\left\{ \begin{array}{l} \text{ALS} \\ \text{ARS} \\ \text{RAL} \\ \text{RAR} \\ \text{ALR} \\ \text{ALF} \\ \text{ERA} \\ \text{ELA} \end{array} \right\}$	comments
label	$\left\{ \begin{array}{l} \text{BLS} \\ \text{BRS} \\ \text{RBL} \\ \text{RBR} \\ \text{BLR} \\ \text{BLF} \\ \text{ERB} \\ \text{ELB} \end{array} \right\}$	[CLE]	[SLB]	,	$\left\{ \begin{array}{l} \text{BLS} \\ \text{BRS} \\ \text{RBL} \\ \text{RBR} \\ \text{BLR} \\ \text{BLF} \\ \text{ERB} \\ \text{ELB} \end{array} \right\}$	comments

CLE, SLA, or SLB appearing alone or in any valid combination with each other are assumed to be a Shift-Rotate machine instruction.

The Shift-Rotate instructions must be given in the order shown. At least one and up to four are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register.

No-Operation Instruction

When a no-operation is encountered in a program, no action takes place; the computer goes on to the next instruction. A full memory cycle is used in executing a no-operation instruction.

label	NOP	comments
-------	-----	----------

A subroutine to be entered by a JSB instruction should have a

NOP as the first statement. The return address can be stored in the location occupied by the NOP during execution of the program. A NOP statement causes the Assembler to generate a word of zeros.

Alter-Skip Group

The Alter-Skip group contains 19 basic instructions that can be combined to produce more than 700 different single cycle operations.

CLA	Clear the A-Register
CLB	Clear the B-Register
CMA	Complement the A-Register
CMB	Complement the B-Register
CCA	Clear, then complement the A-Register (set to ones)
CCB	Clear, then complement the B-Register (set to ones)
CLE	Clear the E-Register
CME	Complement the E-Register
CCE	Clear, then complement the E-Register
SEZ	Skip next instruction if E is zero
SSA	Skip if sign of A is positive (0).
SSB	Skip if sign of B is positive (0).
INA	Increment A by one.
INB	Increment B by one.
SZA	Skip if contents of A equals zero
SZB	Skip if contents of B equals zero
SLA	Skip if least significant bit of A is zero
SLB	Skip if least significant bit of B is zero
RSS	Reverse the sense of the skip instructions. If no skip instructions precede in the statement, skip the next instruction.

These instructions may be combined as follows:

label	$\left\{ \begin{array}{l} \text{CLA} \\ \text{CMA} \\ \text{CCA} \end{array} \right\}$	[, SEZ]	$\left\{ \begin{array}{l} \text{CLE} \\ \text{CME} \\ \text{CCE} \end{array} \right\}$	[, SSA] [, SLA] [, INA] [, SZA] [, RSS]	comments
label	$\left\{ \begin{array}{l} \text{CLB} \\ \text{CMB} \\ \text{CCB} \end{array} \right\}$	[, SEZ]	$\left\{ \begin{array}{l} \text{CLE} \\ \text{CME} \\ \text{CCE} \end{array} \right\}$	[, SSB] [, SLB] [, INB] [, SZB] [, RSS]	comments

The Alter-Skip instructions must be given in the order shown. At least one and up to eight are included in one statement. Instructions referring to the A-register may not be combined in the same statement with those referring to the B-register. When two or more skip functions are combined in a single operation, a skip occurs if any one of the conditions exists. If a word with RSS also includes both SSA and SLA (or SSB and SLB) a skip occurs only when sign and least significant bit are both set (1).

3.3 INPUT/OUTPUT, OVERFLOW, AND HALT

The input/output instructions allow the user to transfer data to and from an external device via a buffer, to enable or disable external interrupt, or to check the status of I/O devices and operations. A subset of these instructions permits checking for an arithmetic overflow condition.

Input/Output instructions require the designation of a select code, sc, which indicates one of 64 input/output channels or functions. Each channel consists of a connect/disconnect control bit, a flag bit, and a buffer of up to 16 bits. The setting of the control bit indicates that a device associated with the channel is operable. The flag bit is set automatically when transmission between the device and the buffer is completed. Instructions are also available to test or clear the flag bit for the particular channel. If the interrupt system is enabled, setting of the flag causes program interrupt to occur; control transfers to the interrupt location related to the channel.

Expressions used to represent select codes (channel numbers) must have a value of less than 2^6 . The value specifies the device or operation referenced. Instructions which transfer data between the A or B register and a buffer, access the Switch register when $sc = 1$. The character C appended to such an instruction clears the overflow bit after the transfer from the Switch register is complete.

Input/Output

Prior to any input/output data transmission, the control bit is set. The instruction which enables the device may also transfer data between the device and the buffer.

label	STC	sc [, C]	comments
-------	-----	----------	----------

Set I/O control bit for channel specified by sc . STC transfers or enables transfer of an element of data from an input device to the buffer or to an output device from the buffer. The exact function of the STC depends on the device; for the 2752A Teleprinter, an STC enables transfer of a series of bits. If $sc = 1$, this statement is treated as NOP. The C option clears the flag bit for the channel.

label	CLC	sc [, C]	comments
-------	-----	----------	----------

Clear I/O control bit for channel specified by sc . When the control bit is cleared, interrupt on the channel is disabled, although the flag may still be set by the device. If $sc = 0$, control bits for all channels are cleared to zero; all devices are disconnected. If $sc = 1$, this statement is treated as NOP.

label	LIA	sc [, C]	comments
-------	-----	----------	----------

Load into A the contents of the I/O buffer indicated by sc .

label	LIB	sc [, C]	comments
-------	-----	----------	----------

Load into B the contents of the I/O buffer indicated by sc .

label	MLA	sc [, C]	comments
-------	-----	----------	----------

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into A.

label	MIB	sc [, C]	comments
-------	-----	----------	----------

Merge (inclusive "or") the contents of the I/O buffer indicated by sc into B.

label	OTA	sc [, C]	comments
-------	-----	----------	----------

Output the contents of A to the I/O buffer indicated by sc.

label	OTB	sc [, C]	comments
-------	-----	----------	----------

Output the contents of B to the I/O buffer indicated by sc.

label	STF	sc	comments
-------	-----	----	----------

Sets the flag bit of the channel indicated by sc. If sc = 0, STF enables the interrupt system. A sc code of 1 causes the overflow bit to be set.

label	CLF	sc	comments
-------	-----	----	----------

Clear the flag bit to zero for the channel indicated by sc. If sc = 0, CLF disables the interrupt system. If sc = 1, the overflow bit is cleared to zero.

label	SFC	sc	comments
-------	-----	----	----------

Skip the next instruction if the flag bit for channel sc is clear. If sc = 1, the overflow bit is tested.

label	SFS	sc	comments
-------	-----	----	----------

Skip the next instruction if the flag bit for channel sc is set. If sc = 1, the overflow is tested.

Overflow

In addition to the use of a select code of 1, the overflow bit may be accessed by the following instructions:

label	CLO	comments
-------	-----	----------

Clear the overflow bit.

label	STO	comments
-------	-----	----------

Set overflow bit.

label	SOC	[C]	comments
-------	-----	-------	----------

Skip the next instruction if the overflow bit is clear. The C option clears the bit after the test is performed.

label	SOS	[C]	comments
-------	-----	-------	----------

Skip the next instruction if the overflow bit is set. The C option clears the bit after the test is performed.

If the C option is not used, the comments portion should be omitted.

Halt

label	HLT	[sc [, C]]	comments
-------	-----	----------------	----------

Halt the computer. The machine instruction word is displayed in the T-Register. If the C option is used, the flag bit associated with channel sc is cleared.

If neither the select code nor the C option is used, the comments portion should also be omitted.

3.4 EXTENDED ARITHMETIC UNIT

The ten instructions available with the Extended Arithmetic Unit increase the overall efficiency of the computer by simultaneously decreasing program run time and increasing available core storage.

label	MPY	$\left\{ \begin{array}{l} m[, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	--	----------

The MPY instruction multiplies the contents of the A-Register by the contents of m. The product is stored in registers B and A. B contains the sign of the product and the 15 most significant bits; A contains the least significant bits.

label	DIV	$\left\{ \begin{array}{l} m[, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	--	----------

The DIV instruction divides the contents of registers B and A by the contents of m. The quotient is stored in A and the remainder in B. Initially B contains the sign and the 15 most significant bits of the dividend; A contains the least significant bits.

label	DLD	$\left\{ \begin{array}{l} m[, I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	--	----------

The DLD instruction loads the contents of locations m and m + 1 into registers A and B, respectively.

label	DST	m[, I]	comments
-------	-----	----------	----------

The DST instruction stores the contents of registers A and B in locations m and m + 1, respectively.

MPY, DIV, DLD, DST results in two machine words: a word for the instruction code and one for the operand.

The above four instructions are available without the Extended Arithmetic Unit option as software subroutines.† As a part of the Extended Arithmetic option, they require less core storage and can be executed in less time.

The following six instructions can be used only on machines with the Extended Arithmetic Unit. These shift-rotate instructions provide the capability to shift or rotate the B- and A-Registers n number of bit positions, where $1 \leq n \leq 16$.

label	ASR	n	comments
-------	-----	---	----------

The ASR instruction arithmetically shifts the B- and A-Registers right n bits. The sign bit (bit 15 of B) is extended.

label	ASL	n	comments
-------	-----	---	----------

The ASL instruction arithmetically shifts the B- and A-Register left n bits. Zeroes are placed in the least significant bits. The sign bit (bit 15 of B) is unaltered. The overflow bit is set if bit 14 differs from bit 15 before each shift, otherwise, exit with Overflow bit cleared.

label	RRR	n	comments
-------	-----	---	----------

The RRR instruction rotates the B- and A-Registers right n bits.

label	RRL	n	comments
-------	-----	---	----------

The RRL instruction rotates the B- and A-Registers left n bits.

label	LSR	n	comments
-------	-----	---	----------

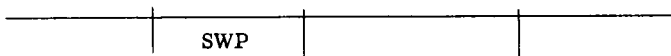
The LSR instruction logically shifts the B- and A-Registers right n bits. Zeroes are placed in the most significant bits.

label	LSL	n	comments
-------	-----	---	----------

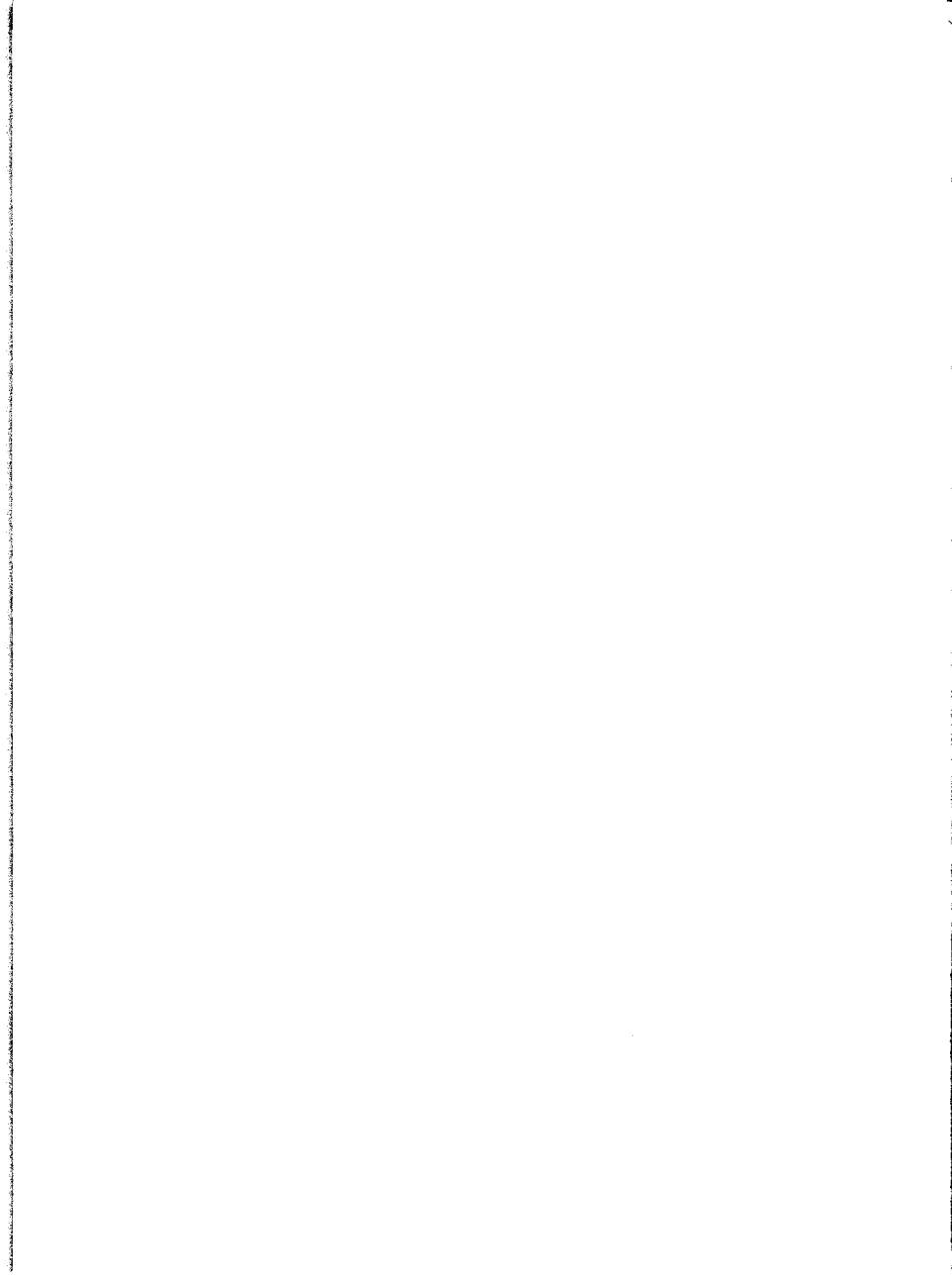
The LSL instruction logically shifts the B- and A-Registers left n bits. Place zeroes into the least significant bits.

† See ARITHMETIC SUBROUTINE CALLS, Section 4.7.

The SWP may be used only in a configuration which includes the Extended Arithmetic Unit option.



Exchange the contents of the A- and B-Registers. The contents of the A-Register are shifted into the B-Register and the contents of the B-Register are shifted into the A-Register.



The pseudo instructions control the Assembler, establish program relocatability, and define program linkage as well as specify various types of constants, blocks of memory, and labels used in the program. With the Extended Assembler, pseudo instructions also control listing output.

4.1 ASSEMBLER CONTROL

The Assembler control pseudo instructions establish and alter the contents of the base page and program location counters, and terminate assembly processing. Labels may be used but they are ignored by the Assembler.

	NAM	[name]	comments
--	-----	--------	----------

NAM defines the name of a relocatable program. A relocatable program must begin with a NAM statement.† A relocatable program is assembled assuming a starting location of zero (i.e., zero relative). The name may be a symbol of one to five alphanumeric characters the first of which must be alphabetic or a period. The program name is printed on the list output. The name is optional and if omitted, the comments must be omitted also.

	ORG	m	comments
--	-----	---	----------

The ORG statement defines the origin of an absolute program, or the origin of subsequent sections of absolute or relocatable programs.

An absolute program must begin with an ORG statement.† The operand m, must be a decimal or octal integer specifying the initial setting of the program location counter.

†The Control Statement, the HED instruction, and comments may appear prior to the NAM or ORG statements. If the Control Statement (ASMB,...) does not appear on tape preceding the program it must be entered from the Teleprinter.

ORG statements may be used elsewhere in the program to define starting addresses for portions of the object code. For absolute programs the Operand field, m, may be any expression. For relocatable programs, m, must be a program relocatable expression; it may not be base page or common relocatable or absolute. An expression is evaluated modulo 2^{15} . Symbols must be previously defined. All instructions following an ORG are assembled at consecutive addresses starting with the value of the operand.

	ORR	comments
--	-----	----------

ORR resets the program location counter to the value existing when an ORG or ORB instruction was encountered.

Example:

	NAM	RSET	SET PLC TO VALUE OF ZERO, ASSIGN
FIRST	ADA		RSET AS NAME OF PROGRAM.
	ADA	CTRL	ASSUME PLC AT FIRST+2280.
	ORG	FIRST+2926	SAVE PLC VALUE OF FIRST+2280 AND SET PLC TO FIRST+2926.
	JMP	EVEN+1	ASSUME PLC AT FIRST+3004
	ORR		RESET PLC TO FIRST+2280.

More than one ORG or ORB statement may occur before an ORR is used. If so, when the ORR is encountered, the program location counter is reset to the value it contained when the first ORG or ORB of the string occurred.

Example:

Label	Operator	Operand	Comments
	NAM	RSET	SET PLC TO ZERO
FIRST	ADA		
	*		
	*		
	LDA	WYZ	ASSUME PLC AT FIRST+2250
	ORG	FIRST+2500	SET PLC TO FIRST+2500
	*		
	*		
	LDB	ERA	ASSUME PLC AT FIRST+2750
	ORG	FIRST+2900	SET PLC TO FIRST+2900
	*		
	*		
	CLE		ASSUME PLC AT FIRST+2920
	ORR		RESET PLC TO FIRST+2250

If a second ORR appears before an intervening ORG or ORB, the second ORR is ignored.

ORB cannot be used to reset the location counter for locations in the base page that are governed by the ORB statement.

ORB	comments
-----	----------

ORB defines the portion of a relocatable program that must be assigned to the base page by the Assembler. The Label field if given is ignored, and the statement requires no operand. All statements that follow the ORB statement are assigned contiguous locations in the base page. Assignment to the base page terminates when the Assembler detects an ORG, ORR, or END statement.

When more than one ORB is used in a program. Each ORB causes the Assembler to resume assigning base page locations at the address following the last assigned base page location.

An ORB statement in an absolute program has no significance and is flagged as an error.

Example:

1	5	10	15	20	25	30	35	40	45	50
		NAM	PROG			ASSIGN ZERO AS RELATIVE STARTING				
		.				LOCATION FOR PROGRAM PROG.				
		.								
		.								
		ORB				ASSIGN ALL FOLLOWING STATEMENTS				
						TO BASE PAGE.				
I	A	R	E	A	B	S	S	1	0	0
		.								
		.								
		ORR				CONTINUE MAIN PROGRAM.				
		.								
		.								
		ORB				RESUME ASSIGNMENT AT NEXT				
		.				AVAILABLE LOCATION IN BASE PAGE.				
		.								
		ORR				CONTINUE MAIN PROGRAM.				

The IFN and IFZ pseudo instructions cause the inclusion of instructions in a program provided that either an "N" or "Z", respectively, is specified as a parameter for the ASMB control statement.† The IFN or IFZ instruction precedes the set of statements that are to be included. The pseudo instruction XIF serves as a terminator. If XIF is omitted, END acts as a terminator to both the set of statements and the assembly. IFN and IFZ may be used only when the source program is translated by the Extended Assembler which is provided for 8K or larger machines.

	IFN	comments
	.	
	.	
	XIF	

All source language statements appearing between the IFN and the XIF pseudo instructions are included in the program if the character "N" is specified on the ASMB control statement.

All source language statements appearing between the IFZ and the XIF pseudo instructions are included in the program if the character "Z" is specified on the ASMB control statement.

	IFZ	comments
	.	
	.	
	XIF	

† See CONTROL STATEMENT, Section 5.1.

When the particular letter is not included on the control statement, the related set of statements appears on the Assembler output listing but is not assembled.

Any number of IFN-XIF and IFZ-XIF sets may appear in a program, however, they may not overlap. An IFZ or IFN intervening between an IFZ or IFN and the XIF terminator results in a diagnostic being issued during compilation; the second pseudo instruction is ignored.

Both IFN-XIF and IFZ-XIF pseudo instructions may be used in the program; however, only one type will be selected in a single assembly. Therefore, if both characters 'N' and 'Z' appear in the control statement, the character which is listed last will determine the set of coding that is to be included in the program.

Example:

Label	5	Operation	15	20	25	30	35	40	Comments	45	50
		NAM	TRAVL								
		.									
		.									
		.									
		IFZ									
		LDA	CAR								
		CMA	SZA								
		JMP	NO.GO								
		LDA	MILES								
		DIV	SPEED								
		STA	GAS								
		XIF									
		.									
		.									
		.									
		IFN									
		LDA	PLANE								
		CMA	SZA								
		JMP	NO.GO								
		LDA	TIME								
		CPA	COST								
		XIF									
NO.GO		HLT	77								
		.									
		.									
		.									
		END									

Program TRAVL will perform computations involving either or neither CAR or PLANE considerations depending on the presence or absence of Z or N parameters in the Control Statement.

Example:

Label	Operation	Operand	Comments
	NAM	WAGE	
	*		
	*		
	*		
	JSB	HOUR	
	MPY	TIME1	
	IFZ		
	JSB	OVTIM	
	MPY	TIME2	
	*		
	*		
	*		
TIME1	DEC	40	
TIME2	BSS	1	
	END		

Program WAGES computes a weekly wage value. Overtime consideration will be included in the program if "Z" is included in the parameters of the Control Statement.

The REP pseudo instruction, available in the Extended Assembler only, causes the repetition of the statement immediately following it a specified number of times.

label	REP	n	comments
-------	-----	---	----------

The statement following the REP in the source program is repeated n times. The n may be any absolute expression. Comment lines (indicated by an asterisk in character position 1) are not repeated by REP. If a comment follows a REP instruction, the comment is ignored and the instruction following the comment is repeated.

A label specified in the REP pseudo instruction is assigned to the first repetition of the statement. A label cannot be part of the instruction to be repeated; it would result in a doubly defined symbol error.

Example:

```
TRIPL    CLA          3
          REP          DATA
          ADA
```

The above source code would generate the following:

	CLA		Clear the A-Register;
	REP		triple the contents of
TRIPL	ADA	DATA	DATA; store the sum
	ADA	DATA	in the A-Register.
	ADA	DATA	

Example:

```
FILL     REP          100B
          NOP
```

The example above loads 100₈ memory locations with the NOP instruction. The first location is labeled FILL.

Example:

```
          REP 2
          *MULTIPLY BY DATA2.
          MPY DATA
```

The above source code would generate the following:

```
          REP
          MPY DATA
          MPY DATA
```

END	[m]	comments
-----	-------	----------

This statement terminates the program; it marks the physical end of the source language statements. The Operand field, m, may contain a name appearing as a statement label in the current program or it may be blank. If a name is entered, it identifies the location to which the BCS loader transfers control after a relocatable program is loaded. A NOP should be stored in this location; the loader transfers control via a JSB.

If the Operand field is blank, the Comments field must be blank also, otherwise, the Assembler attempts to interpret the first five characters of the comments as the transfer address symbol.

The Label field of the END statement is ignored.

4.2 OBJECT PROGRAM LINKAGE

Linking pseudo instructions provide a means for communication between a main program and its subroutines or among several subprograms that are to be run as a single program. These instructions may be used only in a relocatable program.

The Label field of this class is ignored in all cases. The Operand field is usually divided into many subfields, separated by commas. The first space not preceded by a comma or a left parenthesis terminates the entire field.

COM	name ₁ [(size ₁)]	, name ₂ [(size ₂)] , . . . , name _n [(size _n)]	comments
-----	--	---	----------

COM reserves a block of storage locations that may be used in common by several subprograms. Each name identifies a segment of the block for the subprogram in which the COM statement appears. The sizes are the number of words allotted to the related segments. The size is specified as an octal or decimal integer. If the size is omitted, it is assumed to be one.

Any number of COM statements may appear in a subprogram. Storage locations are assigned contiguously; the length of the block is equal to the sum of the lengths of all segments named in all COM statements in the subprogram.

To refer to the common block, other subprograms must also include a COM statement. The segment names and sizes may be the same or they may differ. Regardless of the names and sizes specified in the separate subprograms, there is only one common block for the combined set. It has the same relative origin; the content of the n^{th} word of common storage is the same for all subprograms.

Example:

Label	Operation	Operand	Comment
PROG1	COM	ADDR1(5), ADDR2(10), ADDR3(10)	
	*		
	LDA	ADDR2+1	PICK UP SECOND WORD OF SEGMENT
	*	ADDR2+1	
	*		
	END		
	*		
PROG2	COM	AAA(2), AAB(2), AAC, AAD(20)	
	*		
	LDA	AAD+1	PICK UP SECOND WORD OF SEGMENT
	*	AAD+1	

Organization of common block:

<u>PROG1</u> <u>name</u>	<u>PROG2</u> <u>name</u>	<u>Common</u> <u>Block</u>
ADDR1	AAA	(location 1)
		(location 2)
	AAB	(location 3)
		(location 4)
	AAC	(location 5)
ADDR2	AAD	(location 6)
		(location 7)
		(location 8)
		(location 9)
		(location 10)
		(location 11)
		(location 12)
		(location 13)
		(location 14)
		(location 15)
ADDR3		(location 16)
		(location 17)
		(location 18)
		(location 19)
		(location 20)
		(location 21)
		(location 22)
		(location 23)
		(location 24)
		(location 25)

The LDA instructions in the two subprograms each refer to the same location in common storage, location 7.

The segment names that appear in the COM statements can be used in the Operand fields of DEF, ABS, EQU, or any Memory Reference statement; they may not be used as labels elsewhere in the program.

The loader establishes the origin of the common block; the origin cannot be set by the ORG or ORB pseudo instruction. All references to the common area are relocatable.

Two or more subprograms may declare common blocks which differ in size. The subprogram that defines the largest block must be the first submitted for loading.

ENT	name ₁ [, name ₂ . . . , name _n]	comments
-----	--	----------

ENT defines entry points to the program or subprogram. Each name is a symbol that is assigned as a label for some machine operation in the program. Entry points allow another subprogram to refer to this subprogram. All entry points must be defined in the program.

Symbols appearing in an ENT statement may not also appear in EXT or COM statements in the same subprogram.

The Label field of the ENT instruction is ignored.

EXT	name ₁ [, name ₂ . . . , name _n]	comments
-----	--	----------

This instruction designates labels in other subprograms which are referenced in this subprogram. The symbols must be defined as entry points by the other subprograms.

The symbols defined in the EXT statement may appear in Memory Reference statements, the EQU or DEF pseudo instructions. An external symbol must appear alone; it may not be in a multiple term expression or be specified as indirect in other than a DEF pseudo instruction. References to external locations are processed by the BCS loader as indirect addresses linked through the base page.

Symbols appearing in EXT statements may not also appear in ENT or COM statements in the same subprogram. The label field is ignored.

Example:

1	5	10	15	20	25	30	35	40	45	50
Label	Operation		Operand						Comments	
PROGA	NOP									
	LDA	SAMD								
	.									
	.									
	.									
	JMP	SAND								
	EXT	SAMD, SAND								
	ENT	PROGA								
	END									
	.									
	.									
PROGB	NOP									
	.									
	.									
SAMD	OCT	767								
SAND	STA	SAMD								
	.									
	.									
	ENT	SAMD, SAND								
	.									
	.									
	JSB	PROGA								
	.									
	.									
	EXT	PROGA								
	.									
	.									
	END									

4.3 ADDRESS AND SYMBOL DEFINITION

The pseudo operations in this group assign a value or a word location to a symbol which is used as an operand elsewhere in the program.

label	DEF	m [, I]	comments
-------	-----	-----------	----------

The address definition statement generates one word of memory as a 15-bit address which may be used as the object of an indirect address found elsewhere in the source program. The symbol appearing in the label is that which is referenced; it appears in the Operand field of a Memory Reference instruction.

The operand field of the DEF statement may be any positive expression in an absolute program; in a relocatable program it may be a relocatable expression or an absolute expression with a value of less than 1008. Symbols that do appear in the Operand field, may appear as operands of EXT or COM statements, in the same subprogram and as entry points in other subprograms.

The expression in the Operand field may itself be indirect and make reference to another DEF statement elsewhere in the source program.

Example:

1	Label	5	Operand	10	15	20	25	30	35	40	Comments	45	50
			NAM	PROGN							ZERO-RELATIVE START OF PROGRAM.		
			EXT	SINE, SQR									
			COM	SCMA(20), SCMB(50)									
			.										
			.										
			JSB	SINE							EXECUTE SINE ROUTINE		
			.										
			LDA	XCMA, I							PICK UP COMMON WORD INDIRECTLY.		
			.										
			.										
XCMA		DEF	SCMA								SCMA IS A 15-BIT ADDRESS.		
			.										
			JSB	XSQ, I							GET SQUARE ROOT USING TWO-LEVEL		
XSQ		DEF	XSQR, I								INDIRECT ADDRESSING.		
			.										
			.										
XSQR		DEF	SQR								SQR IS A 15-BIT ADDRESS.		
			END	PROGN									

The DEF statement provides the necessary flexibility to perform address arithmetic in programs which are to be assembled in relocatable form. Relocatable programs should not modify the operand of a memory reference instruction.

In the example below, if TBL and LDTBL are in different pages, the BCS Loader processes TBL as an indirect address linked through the base page. The ISZ erroneously increments the loader provided reference to the base page rather than the value of TBL.

Example:

Label	Operation	Operand	Comments
LDTBL	LDA	TBL	
	ISZ	LDTBL	
TBL	BSS	100	

Assuming the loader might assign absolute locations comparable to the following octal values:

Page	Loc	Opcode	Reference
(0)	(700)	DEF	4000
		.	
		.	
(1)	(200)	LDA	(0) 700(I)
		.	
		.	
(1)	(300)	ISZ	(1) 200
		.	
		.	
(2)	(0)		(TBL)

It can be seen that the ISZ instruction would increment the quantity 700 rather than the address of the table (4000₈).

The following assures correct address modification during program execution.

Example:

Label	Operation	Operand	Comment
ITBL	DEF	TBL	
LDTBL	LDA	ITBL, I	
	.		
	.		
	.		
	ISZ	ITBL	
	.		
	.		
TBL	BSS	100	

This sequence might be stored by the loader as:

<u>Page</u>	<u>Loc</u>	<u>Opcode</u>	<u>Reference</u>
(1)	(200)	DEF	4000
(1)	(201)	LDA	200(I)
		.	
		.	
		.	
(1)	(300)	ISZ	(1) (200)
		.	
		.	
		.	
(2)	(0)		(TBL)

The value of 4000 is incremented; each execution of LDA will access successive locations in the table.

label	ABS	m	comments
-------	-----	---	----------

ABS defines a 16-bit absolute value to be stored at the location represented by the label. The Operand field, m, may be any absolute expression; a single symbol must be defined as absolute elsewhere in the program.

Example:

Label	3	Operand	15	30	45	Comments	60
AB	EQU	35				ASSIGNS THE VALUE OF 35	
						TO THE SYMBOL AB	
M35	ABS	-AB				M35 CONTAINS -35.	
P35	ABS	AB				P35 CONTAINS 35.	
P70	ABS	AB+AB				P70 CONTAINS 70.	
P30	ABS	AB-5				P30 CONTAINS 30.	

label	EQU	m	comments
-------	-----	---	----------

The EQU pseudo operation assigns to a symbol a value other than the one normally assigned by the program location counter. The symbol in the Label field is assigned the value represented by the Operand field. The Operand field may contain any expression. The value of the operand may be common, base page or program relocatable as well as absolute, but it may not be negative. Symbols appearing in the operand must be previously defined in the source program.

The EQU instruction may be used to symbolically equate two locations in memory; or it may be used to give a value to a symbol. The EQU statement does not result in a machine instruction.

Examples:

Label	Operation	Operand	Comment
	NAM	FAM	
	.		
	.		
J3	DEF		
	.		
	.		
	LDA	J3	THE SYMBOLS JFOUR AND J3+1 BOTH
	ADA	ONE	IDENTIFY THE SAME LOCATION. THE
	STA	J3+1	AND OPERATION IS PERFORMED ON
JFOUR	EQU	J3+1	THIS LOCATION.
	.		
	.		
MWH	AND	JFOUR	
	.		
	.		

Examples:

Label	Operation	Operand	Comment
	NAM	STOTB	
	.		
	.		
	COM	TABLA(10)	DEFINES A 10 WORD TABLE, TABLA.
	.		
	.		
TABLB	EQU	TABLA+5	NAMES WORDS 6 THROUGH 10 OF
	.		TABLA AS TABLB.
	.		
	.		
	LDA	TABLB+1	LOADS CONTENTS OF 7TH WORD
	.		COMMON INTO A. THE STATEMENT LDA
	.		TABLA+6 WOULD PERFORM THE SAME
	.		OPERATION
	NAM	REG	
	.		
	.		
A	EQU	0	DEFINES SYMBOL A AS 0 (LOCATION
B	EQU	1	OF A-REGISTER), AND SYMBOL B AS
	.		1 (LOCATION OF B-REGISTER).
	.		
	.		
	LDA	B	LOADS CONTENTS OF B-REGISTER
	.		INTO A-REGISTER.

4.4 CONSTANT DEFINITION

The pseudo instructions in this class enter a string of one or more constant values into consecutive words of the object program. The statements may be named by labels so that other program statements can refer to the fields generated by them.

label	ASC	n, <2n characters>	comments
-------	-----	--------------------	----------

ASC generates a string of 2n alphanumeric characters in ASCII code into n consecutive words.[†] One character is right justified in each eight bits; the most significant bit is zero. n may be any expression resulting in an unsigned decimal value in the range 1 through 28. Symbols used in an expression must be previously defined. Anything in the Operand field following 2n characters is treated as comments. If less than 2n characters are detected before the end-of-statement mark, the remaining characters are assumed to be spaces, and are stored as such. The label represents the address of the first two characters.

Example:

Label	Operator	Operand	Comments							
1	5	15	20	25	30	35	40	45	50	
TTYP	ASC	3,ABCDE								

causes the following:

ALPHABETIC									
	15	14		8	7	6		0	
TTYP	A			B					
	C			D					
	E			A					

EQUIVALENT IN OCTAL NOTATION									
	15	14		8	7	6		0	
TTYP	1	0	1	1	0	2			
	1	0	3	1	0	4			
	1	0	5	0	4	0			

[†] To enter the code for the ASCII symbols which perform some action (e.g., **CR** and **LF**), the OCT pseudo instruction must be used.

label	DEC	$d_1, [d_2, \dots, d_n]$	comments
-------	-----	--------------------------	----------

DEC records a string of decimal constants into consecutive words. The constants may be either integer or real (floating point), and positive or negative. If no sign is specified, positive is assumed. The decimal number is converted to its binary equivalent by the Assembler. The label, if given, serves as the address of the first word occupied by the constant.

A decimal integer must be in the range of 0 to $2^{15} - 1$; it may assume positive, negative, or zero values. It is converted into one binary word and appears as follows:



Examples:

Label	Operator	Operand	Comments
INT	DEC	50, +328, -300	

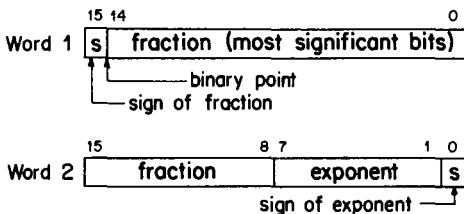
causes the following (octal representation)

	15	14				0
INT	0	0	0	0	6	2
	0	0	0	5	1	0
	1	7	7	3	2	4

A floating point number has two components, a fraction and an exponent. The exponent specifies the power of 10 by which the fraction is multiplied. The fraction is a signed or unsigned number which may be written with or without a decimal point. The exponent is indicated by the letter E and follows a signed or unsigned decimal integer. The floating point number may have any of the following formats:

$\pm n.n \quad \pm n. \quad \pm.n \quad \pm n.nE\pm e \quad \pm.nE\pm e \quad \pm n.E\pm e \quad \pm nE\pm e$

The number is converted to binary, normalized (leading bits differ), and stored in two computer words. If either the fraction or the exponent is negative, that part is stored in two's complement form.



The floating point number is made up of a 7-bit exponent with sign and a 23-bit fraction with sign. The number must be in the approximate range of 10^{-38} through 10^{+38} and zero.

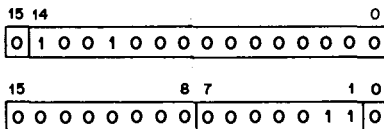
Examples:

1	5	10	15	20	25	30	35	40	45	50
		DEC	.45E1							
		DEC	45.00E-1							
		DEC	4500E-3							
		DEC	4.5							

are all equivalent to

$$.45 \times 10^1$$

and are stored in normalized form as:



Label	Operation	Operand	Comments
	DEC	- .695,400E-4	

are stored as:

1	0	1	0	0	1	1	1	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

0	1	0	1	0	0	0	1	1	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	0	0	0	0	1	0	1	1	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

label	OCT	$o_1 [, o_2 \dots , o_n]$	comments
-------	-----	-----------------------------	----------

OCT stores one or more octal constants in consecutive words of the object program. Each constant consists of one to six octal digits (0 to 177777). If no sign is given, the sign is assumed to be positive. If the sign is negative, the two's complement of the binary equivalent is stored. The constants are separated by commas; the last constant is terminated by a space. If less than six digits are indicated for a constant, the data is right justified in the word. A label, if used, acts as the address of the first constant in the string. The letter B must not be used after the constant in the Operand field; it is significant only when defining an octal term in an instruction other than OCT.

Examples:

Label	Operation	Operand	Comments
	OCT	+0	
	OCT	-2	
NUM	OCT	177,20405,-36	
	OCT	51,77777,-1,10101	
	OCT	107642,177077	
	OCT	1976	ILLEGAL: CONTAINS
	OCT	-177777	DIGIT 9
	OCT	177B	ILLEGAL: CONTAINS
			CHARACTER B

The previous statements are stored as follows:

	15	14				0
	0	0	0	0	0	0
	1	7	7	7	7	6
NUM	0	0	0	1	7	7
	0	2	0	4	0	5
	1	7	7	7	4	2
	0	0	0	0	5	1
	0	7	7	7	7	7
	1	7	7	7	7	7
	0	1	0	1	0	1
	1	0	7	6	4	2
	1	7	7	0	7	7
	X	X	X	X	X	X
	0	0	0	0	0	1
	X	X	X	X	X	X

THE RESULT OF ATTEMPTING TO DEFINE AN ILLEGAL CONSTANT IS UNPREDICTABLE

4.5 STORAGE ALLOCATION

The storage allocation statement reserves a block of memory for data or for a work area.

label	BSS	m	comments
-------	-----	---	----------

The BSS pseudo operation advances the program or base page location counter according to the value of the operand. The Operand field may contain any expression that results in a positive integer. Symbols, if used, must be previously defined in the program. The label, if given, is the name assigned to the storage area and represents the address of the first word. The initial content of the area set aside by the statement is unaltered by the loader.

4.6 ASSEMBLY LISTING CONTROL

Assembly listing control pseudo instructions allow the user to control the assembly listing output during pass 2 or 3 of the assembly process. These pseudo instructions may be used only when the source program is translated by the Extended Assembler provided for 8K or larger machines (8,192-word memory or larger).

	UNL	comments
--	-----	----------

Output is suppressed from the assembly listing, beginning with the UNL pseudo instruction and continuing for all instructions and comments until either an LST or END pseudo instruction is encountered. Diagnostic messages for errors encountered by the Assembler will be printed, however. The source statement sequence numbers (printed in columns 1-4 of the source program listing) are incremented for the instructions skipped.

	LST	comments
--	-----	----------

The LST pseudo instruction causes the source program listing, terminated by a UNL, to be resumed.

A UNL following a UNL, a LST following a LST, and a LST not preceded by a UNL are not considered errors by the Assembler.

	SUP	comments
--	-----	----------

The SUP pseudo instruction suppresses the output of additional code lines from the source program listing. Certain pseudo instructions, because they result in using subroutines, generate more than one line of coding. These additional code lines are suppressed by a SUP instruction until a UNS or the END pseudo instruction is encountered. SUP will suppress additional code lines in the following pseudo instructions:

ASC	DIV	FAD	FSB
OCT	DLN	FDV	MPY
DEC	DST	FMP	

The SUP pseudo instruction may also be used to suppress the listing of literals at the end of the source program listing.

	UNS	comments
--	-----	----------

The UNS pseudo instruction causes the printing of additional coding lines, terminated by a SUP, to be resumed.

A SUP preceded by another SUP, UNS preceded by UNS, or UNS not preceded by a SUP are not considered errors by the Assembler.

	SKP	comments
--	-----	----------

The SKP pseudo instruction causes the source program listing to be skipped to the top of the next page. The SKP instruction is not listed, but the source statement sequence number is incremented for the SKP.

SPC	n	comments
-----	---	----------

The SPC pseudo instruction causes the source program listing to be skipped a specified number of lines. The list output is skipped n lines, or to the bottom of the page, whichever occurs first. The n may be any absolute expression. The SPC instruction is not listed but the source statement sequence number is incremented for the SPC.

label	HED	m	comments
-------	-----	---	----------

The HED pseudo instruction allows the programmer to specify a heading to be printed at the top of each page of the source program listing.

The heading, m, a string of up to 56 ASCII characters, is printed at the top of each page of the source program listing following the occurrence of the HED pseudo instruction. If HED is encountered before the NAM or ORG at the beginning of a program, the heading will be used on the first page of the source program listing. A HED instruction placed elsewhere in the program causes a skip to the top of the next page.

The heading specified in the HED pseudo instruction will be used on every page until it is changed by a succeeding HED instruction.

The source statement containing the HED will not be listed, but source statement sequence number will be incremented.

4.7 ARITHMETIC SUBROUTINE CALLS

The members of this group of pseudo instructions generate calls to arithmetic subroutines external to the source program. These pseudo instructions may be used in relocatable programs only. The Operand field may contain any relocatable expression or an absolute expression resulting in a value of less than 100₈.

label	MPY	$\left\{ \begin{array}{l} m[I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	--	----------

Multiply the contents of the A-register by the contents of m or the quantity defined by the literal and store the product in registers B and A. B contains the sign of the product and the 15 most significant bits; A contains the least significant bits.

label	DIV	$\left\{ \begin{array}{l} m[I] \\ \text{lit} \end{array} \right\}$	comments
-------	-----	--	----------

Divide the contents of registers B and A by the contents of m or the quantity defined by the literal. Store the quotient in A and the remainder in B. Initially B contains the sign and the 15 most significant bits of the dividend; A contains the least significant bits.

label	FMP	$\left\{ \begin{array}{l} m[I] \\ =Fn \end{array} \right\}$	comments
-------	-----	---	----------

Multiply the two-word floating point quantity in registers A and B by the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point product in registers A and B.

label	FDV	$\left\{ \begin{array}{l} m[I] \\ =Fn \end{array} \right\}$	comments
-------	-----	---	----------

Divide the two-word floating point quantity in registers A and B by the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point quotient in A and B.

label	FAD	$\left\{ \begin{array}{l} m[I] \\ =Fn \end{array} \right\}$	comments
-------	-----	---	----------

Add the two-word floating point quantity in registers A and B to the two-word floating point quantity in locations m and m+1 or the quantity defined by the literal. Store the two-word floating point sum in A and B.

label	FSB	$\left\{ \begin{array}{l} m [, I] \\ = F_n \end{array} \right\}$	comments
-------	-----	--	----------

Subtract the two-word floating point quantity in m and m+1 or the quantity defined by the literal from the two-word floating point quantity in registers A and B and store the difference in A and B.

label	DLD	$\left\{ \begin{array}{l} m [, I] \\ = F_n \end{array} \right\}$	comments
-------	-----	--	----------

Load the contents of locations m and m+1 or the quantity defined by the literal into registers A and B respectively.

label	DST	m [, I]	comments
-------	-----	-----------	----------

Store the contents of registers A and B in locations m and m+1 respectively.

Each use of a statement from this group generates two words of instructions. Symbolically, they could be represented as follows:

```

JSB      <. arithmetic pseudo operation>
DEF      m [ , I ]

```

An EXT<. arithmetic pseudo operation> is implied preceding the JSB operation.

In the above operations, the Overflow bit is set when one of the following conditions occurs:

- Integer overflow
- Floating point overflow or underflow
- Division by zero.

Execution of any of the subroutines alters the contents of the E-Register.



The Assembler accepts as input a paper tape containing a control statement and a source language program. A relocatable source language program may be divided into several subroutines; the designation of these elements is optional. The output produced by the Assembler may include a punched paper tape containing the object program, an object program listing, and diagnostic messages.

5.1 CONTROL STATEMENT

The control statement specifies the output to be produced:

ASMB, p_1, p_2, \dots, p_n

"ASMB," is entered in positions 1-5. Following the comma are one or more parameters, in any order, which define the output to be produced. The control statement must be terminated by an end-of-statement mark, **CR** **LF**.

The parameters may be any legal combination of the following starting in position 6:

- A Absolute: The addresses generated by the Assembler are to be interpreted as absolute locations in memory. The program is a complete entity. It may not include NAM, ORB, COM, ENT, EXT, arithmetic pseudo operation statements or literals. The binary output format is that specified for the Basic Binary loader.
- R Relocatable: The program may be located anywhere in memory. Instruction operands are adjusted as necessary. The binary output format is that specified for the BCS Relocating loader.
- B Binary output: A program is to be punched according to one of the above parameters.
- L List output: A program listing is to be produced either during pass two or pass three (if binary output selected) according to one of the above parameters.
- T Table print: List the symbol table at the end of the first pass.

- N Include sets of instructions following the IFN pseudo instruction.
- Z Include sets of instructions following the IFZ pseudo instruction.

Either A or R must be specified in addition to any combination of B, L, or T.

The control statement may be on the same tape as the source program, or on a separate tape; or it may be entered via the Teleprinter keyboard.

5.2 SOURCE PROGRAM

The first statement of the program (other than remarks or a HED statement) must be a NAM statement for a relocatable program or an ORG statement for indicating the origin of an absolute program. The last statement must be an END statement and may contain a transfer address for the start of a relocatable program. Each statement is followed by an end-of-statement mark.

5.3 BINARY OUTPUT

The punch output is defined by the ASMB control statement. The punch output includes the instructions translated from the source program. It does not include system subroutines referenced within the source program (arithmetic subroutine calls, .IOC., .DIO., .ENTR, etc.)

5.4 LIST OUTPUT

Fields of the object program are listed in the following print columns.

<u>Columns</u>	<u>Content</u>
1-4	Source statement sequence number generated by the Assembler
5-6	Blank
7-11	Location (octal)
12	Blank
13-18	Object code word in octal
19	Relocation or external symbol indicator
20	Blank
21-72	First 52 characters of source statement.

Lines consisting entirely of comments (i.e., * in column 1) are printed as follows:

<u>Columns</u>	<u>Content</u>
1-4	Source statement sequence number
5-72	Up to 68 characters of comments

A Symbol Table listing has the following format:

<u>Columns</u>	<u>Content</u>
1-5	Symbol
6	Blank
7	Relocation of external symbol indicator
8	Blank
9-14	Value of the symbol

The characters that designate an external symbol or type of relocation for the Operand field or the symbol are as follows:

<u>Character</u>	<u>Relocation Base</u>
Blank	Absolute
R	Program relocatable
B	Base page relocatable
C	Common relocatable
X	External symbol

At the end of each pass, the following is printed:

```
** NO ERRORS*  
or  
** nnnn ERRORS*
```

The value nnnn, indicates the number of errors.

5.5 OPERATING INSTRUCTIONS

The exact operating procedures for an assembly depend on the available hardware configuration. The user should know the assignment of input/output equipment, † and memory size before initiating an assembly.

† As established when configuring the System Input/Output routines.

One possible allocation of equipment might be as follows:

<u>Assembler Input/Output</u>	<u>Standard Unit Designation</u>	<u>Physical Unit Assignment</u>
Binary Output	Teleprinter Output	2752A Teleprinter
Table Print }	List Output	2753A Tape Punch
List Output }		
Source Program	Input	2737A Punched Tape Reader

Assembly Options

If there are two output devices as shown above, there are only two passes; the Binary and List output are both produced in the second pass. If only one output device is available, the Binary output is produced in the second pass; and the List output, in the third pass.

The Assembler automatically provides a leader and trailer for binary output tapes. To suppress this leader and trailer, set Switch 0 to 1 (up) before the start of Pass 2.

In a three-pass assembly, the diagnostic messages and binary output are written on the same unit. To prevent these messages from being punched on the binary tape (they still appear on the printed output), perform the following steps:

1. Set Switch 15 to 1 (up) before start of Pass 2.
2. When the computer halts with the T-Register containing "102055", turn the punch unit off, and press Run.
3. When the computer again halts with the T-Register containing "102055", turn the punch unit on, and press Run.
4. At the end of Pass 2, set Switch 15 to 0 (down).

Steps 2 and 3 are repeated, each time a diagnostic message is produced.

Operating Procedures: Paper Tape System

The following procedures indicate the sequence of steps for assembly of a source program using the paper tape system.

- A. Set Teleprinter to LINE and check that all equipment to be used is operable.

B. Load the Assembler using the Basic Binary Loader:†

1. Place Assembler binary tape in the device serving as the Standard Input unit (e. g., Punched Tape Reader).
2. Set Switch Register to starting address of Basic Binary Loader (e.g., 007700 for 4K memory, 017700 for 8K memory).
3. Press LOAD ADDRESS.
4. Set Loader switch to ENABLED.
5. Press PRESET.
6. Press RUN.
7. When the computer halts and indicates that the Assembler is loaded (T-Register contains 102077), set Loader switch to PROTECTED.

C. Set Switch Register to starting address of Assembler:

1. If control statement is on tape: 100
2. If control statement is to be entered via Teleprinter: 110

D. Press LOAD ADDRESS.

E. Place source language tape in unit serving as the Standard Input unit (e.g., Punched Tape Reader).

F. Press RUN.

G. If control statement is not on tape (i. e., starting address = 1108), enter it via the Teleprinter, following it by **CR** **LF**.

H. At end of Pass 1 (T-Register contains 102011), the Symbol Table, if requested, is on the Standard List Output unit. To execute Pass 2, replace the source language tape in the Standard Input unit, turn Teleprinter punch unit ON, and press RUN.

I. At the completion of each pass, repeat steps E and F. If a three-pass assembly is being executed, turn Teleprinter punch on at completion of Pass 1 and off at completion of Pass 2.

† The appropriate System Input/Output subroutines (drivers) are assumed to be included with the Assembler.

Operating Procedures: Magnetic Tape System

The following procedures indicate the sequence of steps for assembly of a source program using the Magnetic Tape System.

- A. Load system tape on the Magnetic Tape Unit and check that all equipment to be used is operable.
- B. Load the Assembler using the Magnetic Tape Binary Loader:
 - 1) Set Switch Register to starting address of Magnetic Tape Loader. (For an 8K configuration use 017700.)
 - 2) Press LOAD ADDRESS.
 - 3) Set Assembler identification number in the Switch Register.
 - 4) Press PRESET.
 - 5) Set Loader switch to ENABLED.
 - 6) Press RUN.
 - 7) The computer halts at location 1 with 102077 in the T-Register. The Assembler has been located and loaded. †
 - 8) Set the Loader switch to PROTECTED.
- C. Place source language tape in device serving as the Standard Input unit. (If 100g was specified as starting address of Assembler during Prepare Tape System processing, the control statement must be on tape or 110g must be set in the P-Register.)

†While using the Magnetic Tape System, a halt may occur with the T-Register containing 102011:

<u>A-Register</u>	<u>Explanation</u>	<u>Action</u>
000200	Assembler cannot be located on magnetic tape	Set correct ID number in Switch Register and repeat procedure.
any number not equal to 000200	Tape read error on magnetic tape (timing or parity)	Restart at step B.

D. Press RUN.†

E. If 110g was specified as the starting address of Assembler type in control statement.

F. At the end of assemble, T-Register contains 102077. To assemble another program, return to Step C.

During the operation of the Assembler, the following halts may occur:

<u>T-Register</u>	<u>Explanation</u>	<u>Action</u>
102011	End of first pass.	Return to Step E.
102023	End of second of three passes.	To perform Pass 3, return to Step E. To omit Pass 3 and assemble another program, remove output and return to Step C.
102054	Sense Switch halt on Switch 1.	Press RUN to continue.
102055	Switch 15 option selected to prevent punching of diagnostics on binary output tape.	See preceding instructions.
102057	End of source tape section.	Place next section in unit serving as Standard Input unit and press Run.
102066	Control statement error.	Correct control statement and return to Step E (page 5-5).
102077	End of assembly.	Remove output. To assemble another program, return to Step E (page 5-5).

Several programs may be assembled consecutively without reloading the Assembler. If some of the object programs are to be relocatable and others are to be absolute, the programs that are to be assembled in relocatable form must be processed first. If relocatable program assemblies follow absolute program assemblies, an "R?" error will be diagnosed and the assembler must be reloaded.

† To halt Pass 2 at anytime, set Switch 1 up.

5.6 Object Program Loading

If absolute binary output was specified, the Basic Binary Loader is used to load the object program tape.

If relocatable binary output was specified, the BCS Relocating Loader is used to load the object program tape. If the program refers to other Assembler FORTRAN or ALGOL generated object programs, these tapes are loaded by the Relocating Loader at the same time. If the program refers to .DIO. (the FORTRAN Formatter routine), or if it makes use of Arithmetic pseudo instructions, the Program Library tape must be submitted for loading also.

Listed below are summaries of procedures for normal loading of object programs:†

BASIC BINARY LOADER OPERATING PROCEDURES SUMMARY
A. Place binary object tape in Standard Input unit.
B. Set Switch Register to starting address of Basic Binary Loader (e.g., 007700 for 4K memory, 017700 for 8K memory).
C. Press LOAD ADDRESS.
D. Set Loader switch to ENABLED.
E. Press PRESET.
F. Press RUN.
G. When the computer halts with T-Register containing 102077, set Loader switch to PROTECTED.
H. Set Switch Register to starting address of object program.
I. Press LOAD ADDRESS.
J. Check that all I/O devices are ready and loaded for operation of the program.
K. Press RUN.

†For complete details, see Basic Control System Programmer's Reference Manual.

BASIC CONTROL SYSTEM LOADER OPERATING PROCEDURES SUMMARY

- A. Load the Basic Control System tape using the Basic Binary Loader.
- B. Set Switch Register to 000002, press LOAD ADDRESS, and set Switch Register to 000000.
- C. Place Assembler generated relocatable object tape in Standard Input unit.
- D. Press RUN. The loader types "LOAD" if it expects another relocatable or library program.
- E. If more than one relocatable object tape is to be loaded, repeat Steps C and D for each. Otherwise, set Switch Register to 000004 to load library routines.
- F. Place Program Library tape in device serving as Program Library unit.
- G. Press RUN. When the loading operation is complete, the Loader types "*LST". Press RUN. The Loader types "*RUN" indicating the program is ready for execution.
- H. Press RUN to initiate execution.

5.7 ERROR MESSAGES

Errors detected in the source program are indicated by a 1- or 2-letter mnemonic followed by the sequence number and the first 62 characters of the statement in error. The messages are printed on the list output device during the passes indicated:

<u>Error Code</u>	<u>Pass</u>	<u>Description</u>								
CS	1	Control statement error: a) The control statement contained a parameter other than the legal set. b) Neither A nor R, or both A and R were specified. c) There was no output parameter (B, T or L.)								
DD	1	Doubly defined symbol: A name defined in the symbol table appears more than once as: a) A label of a machine instruction. b) A label of one of the pseudo operations: <table><tr><td>BSS</td><td>EQU</td></tr><tr><td>ASC</td><td>ABS</td></tr><tr><td>DEC</td><td>OCT</td></tr><tr><td>DEF</td><td>Arithmetic subroutine call</td></tr></table> c) A name in the Operand field of a COM or EXT statement. d) A label in an instruction following a REP pseudo operation. e) Any combination of the above. An arithmetic subroutine call symbol appears in a program both as a pseudo instruction and as a label.	BSS	EQU	ASC	ABS	DEC	OCT	DEF	Arithmetic subroutine call
BSS	EQU									
ASC	ABS									
DEC	OCT									
DEF	Arithmetic subroutine call									

Error Code	Pass	Description
EN	1	The symbol specified in an ENT statement has already been defined in an EXT or COM statement.
EN 0000 <symbol>	start of 2	The entry point specified in an ENT statement does not appear in the label field of a machine or BSS instruction. The entry point has been defined in the Operand field of an EXT or COM statement, or has been equated to an absolute value.
IF	1	An IFZ or an IFN follows either an IFZ or an IFN without an intervening XIF. The second pseudo instruction is ignored.
IL	1	Illegal instruction: a) Instruction mnemonic cannot be used with type of assembly requested in control statement. The following are illegal in an absolute assembly: NAM EXT ENT COM ORB Arithmetic sub-routine calls b) The ASMB statement has an R parameter, and NAM has been detected after the first valid Opcode.
IL	2 or 3	Illegal character: A numeric term used in the Operand field contains an illegal character(e.g. an octal constant contains other than +, -, or 0 - 7). Illegal instruction: ORB in an absolute assembly.

<u>Error Code</u>	<u>Pass</u>	<u>Description</u>																																		
M	1, 2 or 3	<p>Illegal operand:</p> <p>a) An operand is missing for an Opcode requiring one.</p> <p>b) Operands are optional and omitted but comments are included for:</p> <p style="margin-left: 100px;">SOC SOS HLT</p> <p>c) An absolute expression in one of the following instructions from a relocatable program is greater than 778.</p> <p style="margin-left: 100px;">Memory Reference DEF Arithmetic subroutine calls</p> <p>d) A negative operand is used with an Opcode field other than ABS, DEC, and OCT.</p> <p>e) A character other than I follows a comma in one of the following statements:</p> <table><tr><td>ISZ</td><td>ADA</td><td>AND</td><td>DEF</td></tr><tr><td>JMP</td><td>ADB</td><td>XOR</td><td>Arithmetic</td></tr><tr><td>JSB</td><td>LDA</td><td>IOR</td><td>Subroutine</td></tr><tr><td></td><td>LDB</td><td>CPA</td><td>calls</td></tr><tr><td></td><td>STA</td><td>CPB</td><td></td></tr><tr><td></td><td>STB</td><td></td><td></td></tr></table> <p>f) A character other than C follows a comma in one of the following statements:</p> <table><tr><td>STC</td><td>MIB</td></tr><tr><td>CLC</td><td>OTA</td></tr><tr><td>LIA</td><td>OTB</td></tr><tr><td>LIB</td><td>HLT</td></tr><tr><td>MIA</td><td></td></tr></table>	ISZ	ADA	AND	DEF	JMP	ADB	XOR	Arithmetic	JSB	LDA	IOR	Subroutine		LDB	CPA	calls		STA	CPB			STB			STC	MIB	CLC	OTA	LIA	OTB	LIB	HLT	MIA	
ISZ	ADA	AND	DEF																																	
JMP	ADB	XOR	Arithmetic																																	
JSB	LDA	IOR	Subroutine																																	
	LDB	CPA	calls																																	
	STA	CPB																																		
	STB																																			
STC	MIB																																			
CLC	OTA																																			
LIA	OTB																																			
LIB	HLT																																			
MIA																																				

Error
Code

Pass

Description

- g) A relocatable expression in the operand field of one of the following:

ABS	ASR	RRL
REP	ASL	LSR
SPC	RRR	LSL

- h) An illegal operator appears in an Operand field (e. g. + or - as the last character).
- i) An ORG statement appearing in a relocatable program includes an expression that is base page or common relocatable or absolute.
- j) A relocatable expression contains a mixture of program, base page, and common relocatable terms.
- k) An external symbol appears in an operand expression or is followed by a comma and the letter I.
- l) The literal or type of literal is illegal for the operation code used (e.g., STA =B7).
- m) An illegal literal code has been used (e.g., LDA =077).
- n) An integer expression in one of the following instructions does not meet the condition $1 \leq n \leq 16$. The integer is evaluated modulo 2^4 .

ASR	RRR	LSR
ASL	RRL	LSL

- o) The value of an 'L' type literal is relocatable.

Error
Code

Pass

Description

- g) A relocatable expression in the operand field of one of the following:

ABS	ASR	RRL
REP	ASL	LSR
SPC	RRR	LSL

- h) An illegal operator appears in an Operand field (e. g. + or - as the last character).

- i) An ORG statement appearing in a relocatable program includes an expression that is base page or common relocatable or absolute.

- j) A relocatable expression contains a mixture of program, base page, and common relocatable terms.

- k) An external symbol appears in an operand expression or is followed by a comma and the letter I.

- l) The literal or type of literal is illegal for the operation code used (e.g., STA =B7).

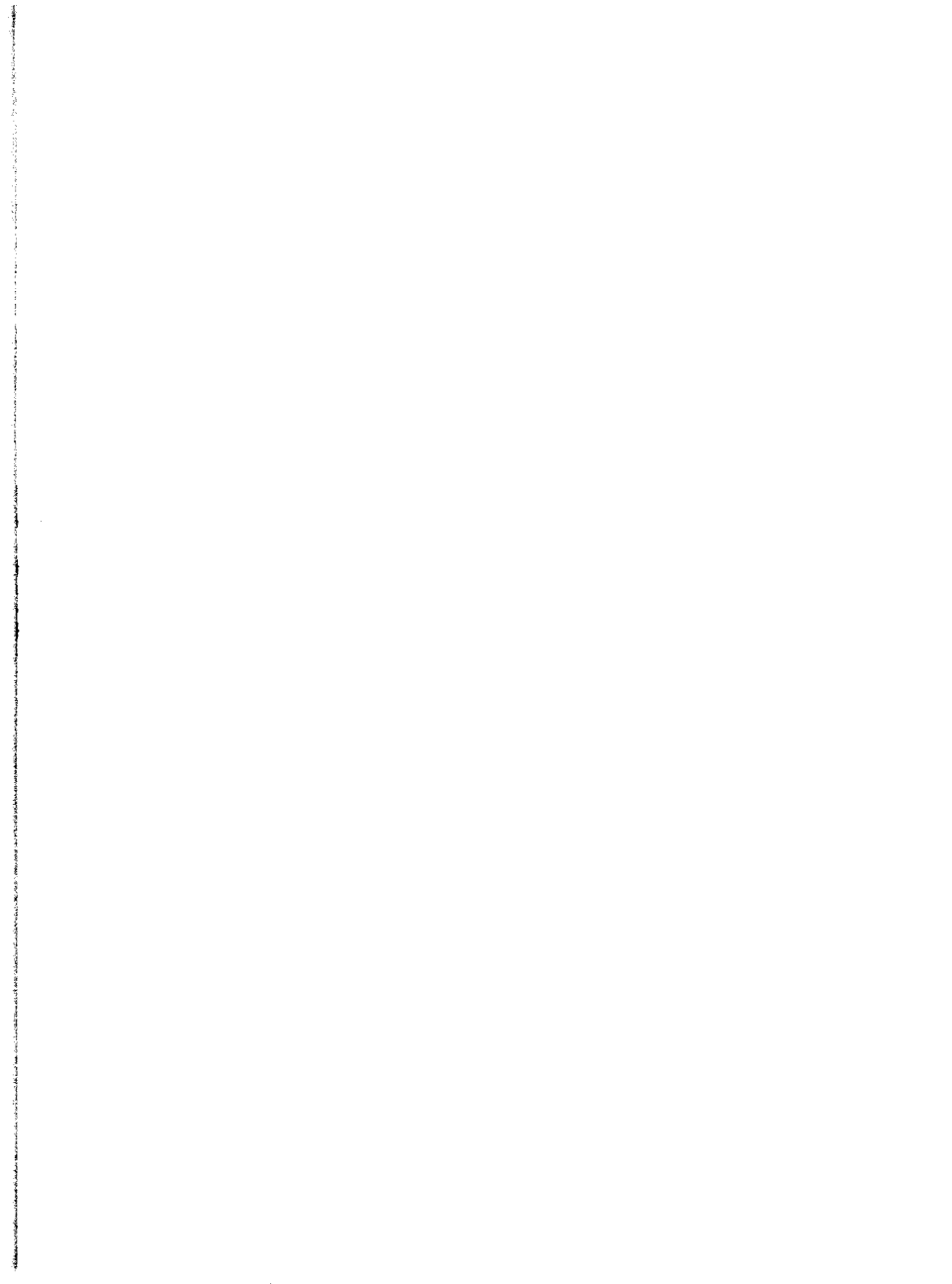
- m) An illegal literal code has been used (e.g., LDA =077).

- n) An integer expression in one of the following instructions does not meet the condition $1 \leq n \leq 16$. The integer is evaluated modulo 2^4 .

ASR	RRR	LSR
ASL	RRL	LSL

- o) The value of an 'L' type literal is relocatable.

<u>Error Code</u>	<u>Pass</u>	<u>Description</u>
SO	1	There are more symbols defined in the program than the symbol table can handle.
SY	1	Illegal Symbol: A Label field contains an illegal character or is greater than 5 characters. A label with illegal characters may result in an erroneous assembly if not corrected. A long label is truncated on the right to 5 characters.
SY	2 or 3	<p>Illegal Symbol: A symbolic term in the Operand field is greater than five characters; the symbol is truncated on the right to 5 characters.</p> <p>Too many control statements: A control statement has been input both on the teleprinter and the source tape or the source tape contains more than one control statement. The Assembler assumes that the source tape control statement is a label, since it begins in column 1. Thus, the commas are considered as illegal characters and the "label" is too long. The binary object tape is not affected by this error, and the control statement entered via the teleprinter is the one used by the Assembler.</p>
TP	1, 2, or 3	An error has occurred while reading or writing magnetic tape; if the T-Register contains 102040, an irrecoverable error has occurred; restart the assembly. Otherwise, correct condition and resume.



BINARY CODED DECIMAL FORMAT

Kennedy 1406/1506 ASCII-BCD Conversion

Symbol	BCD (octal code)	ASCII Equivalent (octal code)	Symbol	BCD (octal code)	ASCII Equivalent (octal code)
(Space)	20	040	A	61	101
!	52	041	B	62	102
#	13	043	C	63	103
\$	53	044	D	64	104
%	34	045	E	65	105
&	60	046	F	66	106
'	14	047	G	67	107
(34	050	H	70	110
)	74	051	I	71	111
*	54	052	J	41	112
+	60	053	K	42	113
,	33	054	L	43	114
-	40	055	M	44	115
.	73	056	N	45	116
/	21	057	O	46	117
0	12	060	P	47	120
1	01	061	Q	50	121
2	02	062	R	51	122
3	03	063	S	22	123
4	04	064	T	23	124
5	05	065	U	24	125
6	06	066	V	25	126
7	07	067	W	26	127
8	10	070	X	27	130
9	11	071	Y	30	131
:	15	072	Z	31	132
;	56	073	[75	133
<	76	074	\	36	134
=	13	075]	55	135
>	16	076			
?	72	077			
@	14	100			

Other symbols which may be represented in ASCII are converted to spaces in BCD (20)

HP 2020A/B ASCII - BCD Conversion

Symbol	ASCII (Octal code)	BCD (Octal code)	Symbol	ASCII (Octal code)	BCD (Octal code)
(Space)	40	20	A	101	61
!	41	52	B	102	62
"	42	37	C	103	63
#	43	13	D	104	64
\$	44	53	E	105	65
%	45	34	F	106	66
&	46	60 †	G	107	67
'	47	36	H	110	70
(50	75	I	111	71
)	51	55	J	112	41
*	52	54	K	113	42
+	53	60	L	114	43
,	54	33	M	115	44
-	55	40	N	116	45
.	56	73	O	117	46
/	57	21	P	120	47
0	60	12	Q	121	50
1	61	01	R	122	51
2	62	02	S	123	22
3	63	03	T	124	23
4	64	04	U	125	24
5	65	05	V	126	25
6	66	06	W	127	26
7	67	07	X	130	27
8	70	10	Y	131	30
9	71	11	Z	132	31
:	72	15	[133	75 †
;	73	56]	135	55 †
<	74	76	^	136	77
=	75	35	_	137	32
>	76	16			
?	77	72			
@	100	14			

† BCD code of 60 always converted to ASCII code 53 (+).

‡ BCD code of 75 always converted to ASCII code 50 (() and
BCD code of 55 always converted to ASCII code 51 ()).



<u>Symbols</u>	<u>Meaning</u>
label	Symbolic label, 1-5 alphanumeric characters and periods
m	Memory location represented by an expression
I	Indirect addressing indicator
C	Clear flag indicator
(m, m+1)	Two-word floating point value in m and m+1
comments	Optional comments
[]	Optional portion of field
{ }	One of set may be selected
P	Program Counter
()	Contents of location
\wedge	Logical product
\vee	Exclusive "or"
\vee	Inclusive "or"
A	A- register
B	B- register
E	E- register
A_n	Bit n of A-register
B_n	Bit n of B-register
b	Bit positions in B- and A-register
(A/B)	Complement of contents of register A or B
(AB)	Two-word floating point value in register A and B
sc	Channel select code represented by an expression
d	Decimal constant
o	Octal constant
r	Repeat count
n	Integer constant
lit	Literal value

MACHINE INSTRUCTIONS

MEMORY REFERENCE

Jump and Increment-Skip

ISZ	m [, I]	(m) + 1 - m; then if (m) = 0, execute P + 2 otherwise execute P + 1
JMP	m [, I]	Jump to m; m - P
JSB	m [, I]	Jump subroutine to m: P + 1 - m; m + 1 - P

Add, Load and Store

ADA	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	(m) + (A) - A
ADB	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	(m) + (B) - B
LDA	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	(m) - A
LDB	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	(m) - B
STA	m [, I]	(A) - m
STB	m [, I]	(B) - m

Logical

AND	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	(m) \wedge (A) - A
XOR	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	(m) \vee (A) - A
IOR	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	(m) \vee (A) - A
CPA	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	If (m) \neq (A), execute P + 2, otherwise execute P + 1
CPB	$\left\{ \begin{matrix} m [, I] \\ \text{lit} \end{matrix} \right\}$	If (m) \neq (B), execute P + 2, otherwise execute P + 1

REGISTER REFERENCE

Shift-Rotate

CLE	0 - E
ALS	Shift (A) left one bit, 0 - A ₀ , A ₁₅ unaltered
BLS	Shift (B) left one bit, 0 - B ₀ , B ₁₅ unaltered
ARS	Shift (A) right one bit, (A ₁₅) - A ₁₄
BRS	Shift (B) right one bit, (B ₁₅) - B ₁₄
RAL	Rotate (A) left one bit
RBL	Rotate (B) left one bit

Shift-Rotate (Continued)

RAR	Rotate (A) right one bit
RBR	Rotate (B) right one bit
ALR	Shift (A) left one bit, $0 - A_{15}$
BLR	Shift (B) left one bit, $0 - B_{15}$
ERA	Rotate E and A right one bit
ERB	Rotate E and B right one bit
ELA	Rotate E and A left one bit
ELB	Rotate E and B left one bit
ALF	Rotate A left four bits
BLF	Rotate B left four bits
SLA	If $(A_0) = 0$, execute $P + 2$, otherwise execute $P + 1$
SLB	If $(B_0) = 0$, execute $P + 2$, otherwise execute $P + 1$

Shift-Rotate instructions can be combined as follows:

$\left\{ \begin{array}{c} \text{ALS} \\ \text{ARS} \\ \text{RAL} \\ \text{RAR} \\ \text{ALR} \\ \text{ALF} \\ \text{ERA} \\ \text{ELA} \end{array} \right\}$	[,CLE] [,SLA]	$\left\{ \begin{array}{c} \text{ALS} \\ \text{ARS} \\ \text{RAL} \\ \text{RAR} \\ \text{ALR} \\ \text{ALF} \\ \text{ERA} \\ \text{ELA} \end{array} \right\}$
$\left\{ \begin{array}{c} \text{BLS} \\ \text{BRS} \\ \text{RBL} \\ \text{RBR} \\ \text{BLR} \\ \text{BLF} \\ \text{ERB} \\ \text{ELB} \end{array} \right\}$	[,CLE] [,SLB]	$\left\{ \begin{array}{c} \text{BLS} \\ \text{BRS} \\ \text{RBL} \\ \text{RBR} \\ \text{BLR} \\ \text{BLF} \\ \text{ERB} \\ \text{ELB} \end{array} \right\}$

No-operation

NOP

Execute $P + 1$

Alter-Skip

CLA	0's - A
CLB	0's - B
CMA	$\overline{(A)}$ - A
CMB	$\overline{(B)}$ - B
CCA	1's - A
CCB	1's - B
CLE	0 - E
CME	$\overline{(E)}$ - E

Alter-Skip (Continued)

CCE	$1 \rightarrow E$
SEZ	If $(E) = 0$, execute $P + 2$, otherwise execute $P + 1$
SSA	If $(A_{16}) = 0$, execute $P + 2$, otherwise execute $P + 1$
SSB	If $(B_{16}) = 0$, execute $P + 2$, otherwise execute $P + 1$
INA	$(A) + 1 \rightarrow A$
INB	$(B) + 1 \rightarrow B$
SZA	If $(A) = 0$, execute $P + 2$, otherwise execute $P + 1$
SZB	If $(B) = 0$, execute $P + 2$, otherwise execute $P + 1$
SLA	If $(A_0) = 0$, execute $P + 2$, otherwise execute $P + 1$
SLB	If $(B_0) = 0$, execute $P + 2$, otherwise execute $P + 1$
RSS	Reverse sense of skip instructions. If no skip instructions precede, execute $P + 2$

Alter-Skip instructions can be combined as follows:

$\left\{ \begin{matrix} \text{CLA} \\ \text{CMA} \\ \text{CCA} \end{matrix} \right\}$	[, SEZ]	$\left\{ \begin{matrix} \text{CLE} \\ \text{CME} \\ \text{CCE} \end{matrix} \right\}$	[, SSA] [, SLA] [, INA] [, SZA] [, RSS]
$\left\{ \begin{matrix} \text{CLB} \\ \text{CMB} \\ \text{CCB} \end{matrix} \right\}$	[, SEZ]	$\left\{ \begin{matrix} \text{CLE} \\ \text{CME} \\ \text{CCE} \end{matrix} \right\}$	[, SSB] [, SLB] [, INB] [, SZB] [, RSS]

INPUT/OUTPUT, OVERFLOW, and HALT

Input/Output

STC	sc [, C]	Set control bit _{sc} , enable transfer of one element of data between device _{sc} and buffer _{sc}
CLC	sc [, C]	Clear control bit _{sc} . If $sc = 0$ clear all control bits
LIA	sc [, C]	$(\text{buffer}_{sc}) \leftarrow A$
LIB	sc [, C]	$(\text{buffer}_{sc}) \leftarrow B$
MIA	sc [, C]	$(\text{buffer}_{sc}) \vee (A) \leftarrow A$
MIB	sc [, C]	$(\text{buffer}_{sc}) \vee (B) \leftarrow B$
OTA	sc [, C]	$(A) \leftarrow \text{buffer}_{sc}$
OTB	sc [, C]	$(B) \leftarrow \text{buffer}_{sc}$
STF	sc	Set flag bit _{sc} . If $sc = 0$, enable interrupt system. $sc = 1$ sets overflow bit.
CLF	sc	Clear flag bit _{sc} . If $sc = 0$, disable interrupt system. If $sc = 1$, clear overflow bit.
SFC	sc	If $(\text{flag bit}_{sc}) = 0$, execute $P + 2$, otherwise execute $P + 1$. If $sc = 1$, test overflow bit.
SFS	sc	If $(\text{flag bit}_{sc}) = 1$, execute $P + 2$, otherwise execute $P + 1$. If $sc = 1$, test overflow bit.

Overflow

CLO		0 → overflow bit
STO		1 → overflow bit
SOC	[C]	If (overflow bit) = 0, execute P + 2, otherwise execute P + 1
SOS	[C]	If (overflow bit) = 0, execute P + 2, otherwise execute P + 1

Halt

HLT [sc [, C]] Halt computer

EXTENDED ARITHMETIC UNIT

MPY	$\left\{ \begin{smallmatrix} m[,1] \\ \text{lit} \end{smallmatrix} \right\}$	(A) \times (m) → (B _{±msb} and A _{lsb})
DIV	$\left\{ \begin{smallmatrix} m[,1] \\ \text{lit} \end{smallmatrix} \right\}$	(B _{±msb} and A _{lsb}) / (m) → A, remainder → B
DLD	$\left\{ \begin{smallmatrix} m[,1] \\ \text{lit} \end{smallmatrix} \right\}$	(m) and (m + 1) → A and B
DST	$\left\{ \begin{smallmatrix} m[,1] \\ \text{lit} \end{smallmatrix} \right\}$	(A) and (B) → m and m + 1
ASR	b	Arithmetically shift (BA) right b bits, B ₁₅ extended
ASL	b	Arithmetically shift (BA) left b bits, B ₁₅ unaltered, 0's to A _{lsb}
RRR	b	Rotate (BA) right b bits
RRL	b	Rotate (BA) left b bits
LSR	b	Logically shift (BA) right b bits, 0's to B _{msb}
LSL	b	Logically shift (BA) left b bits, 0's to A _{lsb}
SWP		(A) → B; (B) → A

PSEUDO INSTRUCTIONS

ASSEMBLER CONTROL

NAM	[name]	Specifies relocatable program and its name.
ORG	m	Gives absolute program origin or origin for a segment of relocatable or absolute program.
ORR		Reset main program location counter at value existing when first ORG or ORB of a string was encountered.
ORB		Defines base page portion of relocatable program.
END	[m]	Terminates source language program. Produces transfer to program starting location, m, if given.
REP	r	
<statement>		Repeat immediately following statement r times.
IFN		
<statements>		Include statements in program if control statement contains N.
XIF		
IFZ		
<statements>		Include statements in program if control statement contains Z.
XIF		

OBJECT PROGRAM LINKAGE

COM	name ₁ [(size ₁)] [, name ₂ [(size ₂)] , . . . , name _n [(size _n)]	Reserves a block of common storage locations. name _i identifies segments of block, each of length size _i .
ENT	name ₁ [, name ₂ , . . . , name _n]	Defines entry points, name _i , that may be referred to by other programs
EX1	name ₁ [, name ₂ , . . . , name _n]	Defines external locations, name _i , which are labels of other programs, referenced by this program.

ADDRESS AND SYMBOL DEFINITION

label	DEF	m[,I]	Generates a 15-bit address which may be referenced indirectly through the label.
label	ABS	m	Defines a 16-bit absolute value to be referenced by the label.
label	EQU	m	Equates the value, m, to the label.

CONSTANT DEFINITION

ASC n, <2n characters>

Generates a string of 2n ASCII characters.

DEC d₁ [, d₂, . . . , d_n]

Records a string of decimal constants of the form:

Integer: ±n

Floating point: ±n.n, ±n., ±.n, ±nE±e,
±n.E±e, ±.nE±e ±n.nE±e,

OCT o₁ [, o₂, . . . , o_n]

Records a string of octal constants of the form:
±000000

STORAGE ALLOCATION

label

BSS

m

Reserves a storage area of length, m, and
identified by the label.

ARITHMETIC SUBROUTINE CALLS

MPY {m[,I]
lit}

(A) x (m) - (B_{±mab} and A_{±ab})

DIV† {m[,I]
lit}

(B_{±mab} and A_{±ab})/(m) - A, remainder - B

FMP {m[,I]
lit}

(AB) x (m, m + 1) - AB

FDV {m[,I]
lit}

(AB)/(m, m + 1) - AB

FAD {m[,I]
lit}

(m, m + 1) + (AB) - AB

FSB {m[,I]
lit}

(AB) - (m, m + 1) - AB

DLD† {m[,I]
lit}

(m) and (m + 1) - A and B

DST† m[,I]

(A) and (B) - m and m + 1

SWPI

(A) - B; (B) - A

ASSEMBLY LISTING CONTROL

UNL

Suppress assembly listing output.

LST

Resume assembly listing output.

SKP

Skip listing to top of next page.

SPC

n

Skip n lines on listing

SUP

Suppress listing of extended code lines
(e.g., as produced by subroutine calls).

UNS

Resume listing of extended code lines.

HED <heading>

Print <heading> at top of each page,
where <heading> is up to 56 ASCII characters.

† For configurations including Extended Arithmetic Unit, these mnemonic generate hardware instructions

† Available only for configurations including Extended Arithmetic Unit

ALPHABETIC LIST OF INSTRUCTIONS

C

ABS	Define absolute value
ADA	Add to A
ADB	Add to B
ALF	Rotate A left 4
ALR	Shift A left 1, clear sign
ALS	Shift A left 1
AND	"And" to A
ARS	Shift A right 1, sign carry
ASC	Generate ASCII characters
ASL	Arithmetic long shift left
ASR	Arithmetic long shift right
BLF	Rotate B left 4
BLR	Shift B left 1, clear sign
BLS	Shift B left 1
BRS	Shift B right 1, carry sign
BSS	Reserve block of storage starting at symbol
CCA	Clear and complement A (1's)
CCB	Clear and complement B (1's)
CCE	Clear and complement E (set E = 1)
CLA	Clear A
CLB	Clear B
CLC	Clear I/O control bit
CLE	Clear E
CLF	Clear I/O flag
CLO	Clear overflow bit
CMA	Complement A
CMB	Complement B

CME	Complement E
COM	Reserve block of common storage
CPA	Compare to A, skip if unequal
CPB	Compare to B, skip if unequal
DEC	Defines decimal constants
DEF	Defines address
DIV	Divide
DLD	Double load
DST	Double store
ELA	Rotate E and A left 1
ELB	Rotate E and B left 1
END	Terminate program
ENT	Entry point
ERA	Rotate E and A right 1
ERB	Rotate E and B right 1
EQU	Equate symbol
EXT	External reference
FAD	Floating add
FDV	Floating divide
FMP	Floating multiply
FSB	Floating subtract
HED	Print heading at top of each page
HLT	Halt
IFN	When N appears in Control Statement, assemble ensuing instructions
IFZ	When Z appears in Control Statement, assemble ensuing instructions
INA	Increment A by 1
INB	Increment B by 1
IOR	Inclusive "or" to A
ISZ	Increment, then skip if zero
JMP	Jump

JSB	Jump to subroutine
LDA	Load into A
LDB	Load into B
LIA	Load into A from I/O channel
LIB	Load into B from I/O channel
LSL	Logical long shift left
LSR	Logical long shift right
LST	Resume list output (follows a UNL)
MIA	Merge (or) into A from I/O channel
MIB	Merge (or) into B from I/O channel
MPY	Multiply
NAM	Names relocatable program
NOP	No operation
OCT	Defines octal constant
ORB	Establish origin in base page
ORG	Establish program origin
ORR	Reset program location counter
OTA	Output from A to I/O channel
OTB	Output from B to I/O channel
RAL	Rotate A left 1
RAR	Rotate A right 1
RBL	Rotate B left 1
RBR	Rotate B right 1
REP	Repeat next statement
RRL	Rotate A and B left
RRR	Rotate A and B right
RSS	Reverse skip sense
SEZ	Skip if E = 0
SFC	Skip if I/O flag = 0 (clear)
SFS	Skip if I/O flag = 1 (set)
SKP	Skip to top of next page

SLA	Skip if LSB of A = 0
SLB	Skip if LSB of B = 0
SOC	Skip if overflow bit = 0 (clear)
SOS	Skip if overflow bit = 1 (set)
SPC	Space n lines
SSA	Skip if sign A = 0
SSB	Skip if sign B = 0
STA	Store A
STB	Store B
STC	Set I/O control bit
STF	Set I/O flag
STO	Set overflow bit
SUP	Suppress list output of additional code lines
SWP	Switch the (A) and (B)
SZA	Skip if A = 0
SZB	Skip if B = 0
UNL	Suppress list output
UNS	Resume list output of additional code lines
XIF	Terminate an IFN or IFZ group of instructions
XOR	Exclusive "or" to A

Following are two sample problems, the second of which implements several options of the Extended Assembler.

A.

PARTS FILE UPDATE

A master file of parts is updated by a parts usage list to produce a new master parts file. A report, consisting of the parts used and their cost, is also produced.

The master file and the parts usage file contain four word records. Each record of the cost report is eleven words long.

The organization of the files is as follows:

Parts Master Files (PRTSM)

Identification	Quantity	Cost/ Item
----------------	----------	---------------

Identification field of the Parts Master Files exists in ASCII although the entire record is read and written in binary.

Parts Usage File (PRTSU)

Identification	Quantity
----------------	----------

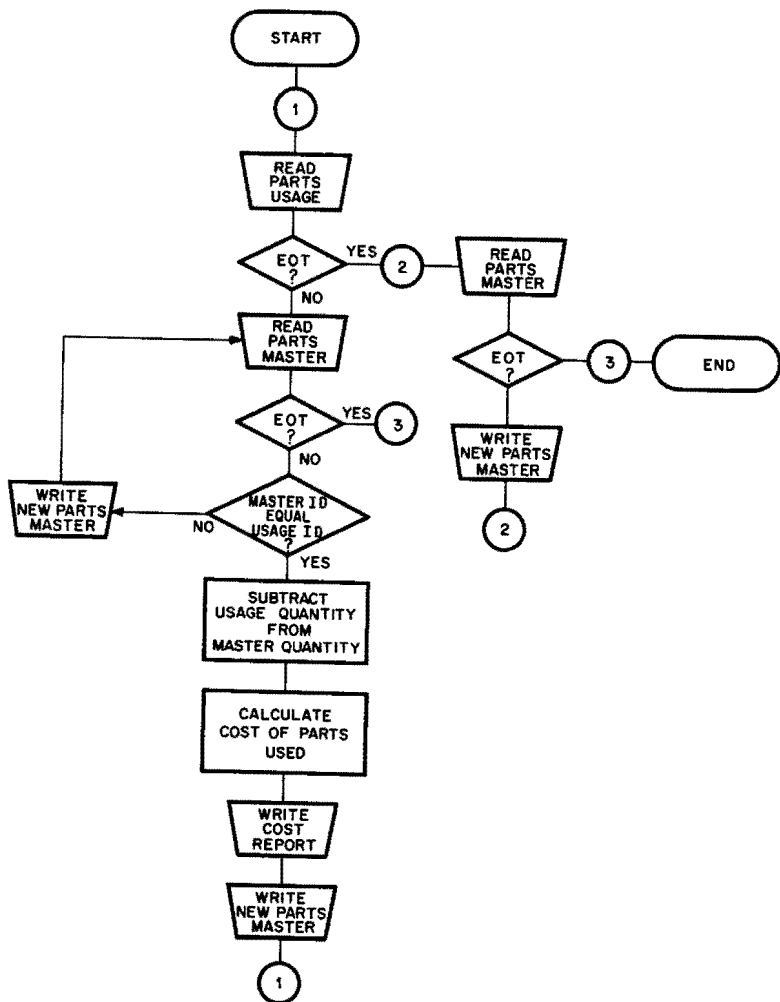
The parts usage file has been recorded in ASCII.

Parts Cost Report (PRTSC)

Identification		Quantity used			Cost for Quantity
----------------	--	---------------	--	--	----------------------

The Parts Cost Report is recorded in ASCII with spacing and editing for printing.

The sample program reads and writes the files, adjusts the new stock levels, and calculates the cost. External subprograms perform the binary-to-decimal and decimal-to-binary conversions and handle unrecoverable input/output errors, invalid data conditions, and normal program termination. Input/output operations are performed using the Basic Control System input/output sub-routine, .IOC.



SAMPLE PROGRAM
GENERAL FLOW CHART

SAMPLE ASSEMBLER SYMBOL TABLE OUTPUT

PAGE 0001

```

                                0001      ASMB,R,B,L,T
START R 000000
PRISM B 000000
PRTSU B 000004
PRTSC B 000010
EOTS1 B 000023
EOTS2 B 000024
MTEMP B 000025
UTEMP B 000026
SWTMP B 000027
SPACS B 000031
DLRSG B 000033
A      000000
B      000001
+IOC. X 000001
BCONV X 000002
DCONV X 000003
ABORT X 000004
HALT X 000005
DT0BI C 000000
DT0BO C 000002
BT0DI C 000003
BT0DO C 000005
OPEN R 000002
SPCFL R 000003
DLD X 000006
DST X 000007
READU R 000013
CKSTU R 000020
RJCTU R 000035
EOTU R 000040
MSGU R 000051
READM R 000063
CKSTM R 000070
RJCTM R 000105
EOTM R 000110
MSGM R 000117
HLTSW R 000137
COMPR R 000140
PROCM R 000157
PROCC R 000165
MPY X 000010
CONVM R 000213
CONU1 R 000224
CONU2 R 000235
CONVC R 000246
WRITC R 000261
CKSTC R 000266
RJCTC R 000276
WRITN R 000301
CKSTN R 000306
RJCTN R 000316
** NO ERRORS*
```

SAMPLE ASSEMBLER LIST OUTPUT

PAGE 0002

0001	00000		NAM	UPDTE	
0002	00000	000000	START	NOP	
0003	00001	026002R		JMP	OPEN
0004	00000			ORB	
0005	00000	000000	PRISM	BSS	4
0006	00004	000000	PRTSU	BSS	4
0007	00010	000000	PRTSC	BSS	11
0008	00023	026063R	EOTS1	JMP	READM
0009	00024	026301R	EOTS2	JMP	WRITN
0010	00025	000000	MTEMP	BSS	1
0011	00026	000000	UTEMP	BSS	1
0012	00027	000000	SWTMP	BSS	2
0013	00031	020040	SPACS	ASC	2,
	00032	020040			
0014	00033	020044	DLRSG	ASC	1, 5
0015	00000		A	EQU	0
0016	00001		B	EQU	1
0017				EXT	.IOC.
0018*					
0019				EXT	BCONV
0020*					
0021				EXT	DCONV
0022*					
0023*					
0024				EXT	ABORT
0025*					
0026*					
0027				EXT	HALT
0028				COM	DT0BI(2),DT0B0,BT0DI(2),BT0D0(2)
0029*					
0030*					
0031*					
0032	00002			ORR	
0033*					
0034	00002	000000	OPEN	NOP	
0035	00003	016006X	SPCFL	DLN	SPACS
	00004	000031B			
0036	00005	016007X		DST	PRTSC+2
	00006	000012B			
0037	00007	016007X		DST	PRTSC+6
	00010	000016B			
0038	00011	060033B	LDA	DLRSG	
0039	00012	070020B	STA	PRTSC+8	
0040	00013	016001X	READU	JSB	.IOC.
0041	00014	010001	OCT	10001	
0042	00015	026035R	JMP	RJCTU	
0043	00016	000004B	DEF	PRTSU	
0044	00017	000004	DEC	4	
0045	00020	016001X	CKSTU	JSB	.IOC.
0046	00021	040001	OCT	40001	
0047	00022	002020	SSA		
0048	00023	026020R	JMP	CKSTU	
0049	00024	001200	RAL		
0050	00025	002020	SSA		
0051	00026	026030R	JMP	**2	
0052	00027	026063R	JMP	READM	
0053*					

ASSIGN STORAGE & CONSTANTS TO BP
MASTER PARTS FILE - BINARY.
PARTS USAGE LIST - ASCII.
PARTS COST REPORT - ASCII.

PERFORM I/O OPERATIONS USING BCS
I/O CONTROL ROUTINE.
ENTRY POINT FOR DECIMAL(ASCII)
TO BINARY CONVERSION SUBPROGRAM.
ENTRY POINT FOR BINARY TO
DECIMAL(ASCII) CONVERSION SUB-
PROGRAM.
ENTRY POINT FOR SUBPROGRAM WHICH
HANDLES UNRECOVERABLE I/O ERRORS
OR INVALID DATA.
END OF PROGRAM SUBROUTINE.
COMMON STORAGE LOCATIONS USED TO
PASS DATA BETWEEN MAIN PROGRAM
AND CONVERSION SUBPROGRAMS.
RESETS PLC AFTER USE OF ORB AT
BEGINNING OF PROGRAM.

STORES EDITING CHARACTERS IN
OUTPUT AREA FOR PARTS COST
REPORT.

READ ONE RECORD FROM USAGE LIST
LOCATED ON STANDARD UNIT 1
(TELEPRINTER INPUT). PRTSU IS
ADDRESS OF STORAGE AREA; AREA IS
4 WORDS LONG.
CHECK STATUS OF UNIT 1.

IF BUSY, LOOP UNTIL FREE.

IF COMPLETE, TRANSFER TO SECTION
WHICH READS MASTER FILE RECORD.

0054	00030	001727	ALF,ALF	TEST END OF TAPE STATUS BIT
0055	00031	001200	RAL	(ORIGINAL BIT 05).
0056	00032	002020	SSA	
0057	00033	026040R	JMP EOTU	IF SET, GO TO EOT PROCEDURE.
0058	00034	026004X	JMP ABORT	IF NOT SET, SOME ERROR CONDITION
0059*				(UNRECOVERABLE) EXISTS.
0060	00035	006020	RJCTU SSB	CHECK CAUSE OF REJECT. IF UNIT
0061	00036	026013R	JMP READU	BUSY LOOP UNTIL FREE. ANY OTHER
0062	00037	026004X	JMP ABORT	CAUSE IS UNRECOVERABLE ERROR.
0063	00040	060023B	EOTU LDA EOTS1	IF END OF USAGE FILE, ALTER
0064	00041	072002R	STA OPEN	PROGRAM SEQUENCE TO BYPASS
0065	00042	060024B	LDA EOTS2	SECTIONS THAT READ AND PROCESS
0066	00043	072140R	STA COMPR	USAGE FILE. PRINT MESSAGE ON
0067	00044	016001X	JSB .IOC.	TELEPRINTER INDICATING EOT.
0068	00045	020002	OCT 20002	
0069	00046	026044R	JMP EOTU+4	
0070	00047	000051R	DEF MSGU	
0071	00050	000011	DEC 9	
0072	00051	042516	MSGU ASC 9,END OF USAGE FILE	
	00052	042040		
	00053	047506		
	00054	020125		
	00055	051501		
	00056	043505		
	00057	020106		
	00060	044514		
	00061	042440		
0073	00062	026063R	JMP READM	
0074	00063	016001X	READM JSB .IOC.	READ A RECORD FROM MASTER PARTS
0075	00064	010105	OCT 10105	FILE ON STANDARD UNIT 05(PUNCHED
0076	00065	026105R	JMP RJCTM	TAPE READER). PRISM IS ADDRESS
0077	00066	000000B	DEF PRISM	OF STORAGE AREA; AREA IS 4 WORDS
0078	00067	000004	DEC 4	LONG. RECORD IS IN BINARY FORMAT
0079	00070	016001X	CKSTM JSB .IOC.	CHECK STATUS OF UNIT 5.
0080	00071	040005	OCT 40005	
0081	00072	002020	SSA	
0082	00073	026070R	JMP CKSTM	IF BUSY, LOOP UNTIL FREE.
0083	00074	001200	RAL	
0084	00075	002020	SSA	
0085	00076	026100R	JMP *+2	
0086	00077	026140R	JMP COMPR	IF COMPLETE, TRANSFER TO EITHER
0087	00100	001727	ALF,ALF	PROCESSING OR WRITE OUTPUT
0088	00101	001200	RAL	DEPENDING ON SETTING OF COMPR.
0089	00102	002020	SSA	TEST FOR END OF TAPE.
0090	00103	026110R	JMP EOTM	IF END, GO TO EOT PROCEDURE.
0091	00104	026004X	JMP ABORT	IF NOT, AN UNRECOVERABLE ERROR
0092*				EXISTS.
0093	00105	006020	RJCTM SSB	CHECK CONTENTS OF B FOR CAUSE OF
0094	00106	026063R	JMP READM	REJECT. IF UNIT BUSY, LOOP UNTIL
0095	00107	026004X	JMP ABORT	FREE. OTHERWISE I/O ERROR EXISTS
0096	00110	062137R	EOTM LDA HLTSW	ALTER PROGRAM SEQUENCE TO HALT
0097	00111	072315R	STA CKSTN+7	EXECUTION AFTER LAST RECORD IS
0098	00112	016001X	JSB .IOC.	WRITTEN PRINT MESSAGE
0099	00113	020002	OCT 20002	INDICATING END OF MASTER INPUT.
0100	00114	026112R	JMP EOTM+2	
0101	00115	000117R	DEF MSGM	
0102	00116	000017	DEC 15	
0103	00117	042516	MSGM ASC 15,END OF MASTER PARTS FILE INPUT	

```

00120 042040
00121 047506
00122 020115
00123 040523
00124 052105
00125 051040
00126 050101
00127 051124
00130 051440
00131 043111
00132 046105
00133 020111
00134 047120
00135 052524
0104 00136 026140R JMP COMPR
0105 00137 026005X HLTSW JMP HALT
0106 00140 000000 COMPR NOP
0107 00141 016224R JSB CONU1
0108 00142 016213R JSB CONVM
0109 00143 060026B LDA UTEMP
0110 00144 064025B LDB MTEMP
0111 00145 050001 CPA B
0112 00146 026157R JMP PROCM
0113 00147 007004 CMB,INB
0114 00150 040001 ADA B
0115 00151 002020 SSA
0116 00152 026004X JMP ABORT
0117 00153 062156R LDA **3
0118 00154 072315R STA CKSTN+7
0119 00155 026301R JMP WRITN
0120 00156 026063R JMP READM
0121 00157 016235R PROCM JSB CONU2
0122 00160 060002B LDA PRISM+2
0123 00161 064027B LDB UTEMP+1
0124 00162 007004 CMB,INB
0125 00163 040001 ADA B
0126 00164 070002B STA PRISM+2
0127 00165 016006X PROCC DLD PRISU
0128 00167 016007X DST PRISC
0129 00171 016006X DLD PRISU+2
0130 00173 016007X DST PRISC+4
0131 00175 060003B LDA PRISM+3
0132 00176 016010X MPY UTEMP+1
0133 00200 070030B STA SWTMP+1
0134 00201 074027B STB SWTMP
0135 00202 016246R JSB CONVC
0136 00203 016006X DLD SWTMP
0137 00205 016007X DST PRISC+9
0138 00207 062212R LDA **3
0139 00210 072315R STA CKSTN+7
0140 00211 026261R JMP WRITC

```

END OF PROGRAM SUBROUTINE.

CONVERT ID NUMBER FIELDS OF
MASTER AND USAGE FILES TO BIN.
LOAD THESE FIELDS FROM TEMPORARY
STORAGE.

COMPARE
IF EQUAL, JUMP TO PROCESSING
IF ID NUMBER OF MASTER GREATER
THAN ID NUMBER OF USAGE, DATA IN
USAGE FILE ERRONEOUS. TERMINATE
RUN.

IF ID MASTER LESS THAN ID USAGE,
ALTER SEQUENCE: READ NEXT MASTER
RECORD IMMEDIATELY AFTER WRITING
CURRENT MASTER RECORD.
CONVERT QUANTITY FIELD OF USAGE
FILE TO BINARY AND SUBTRACT FROM
QUANTITY FIELD OF MASTER AND
STORE RESULT.

STORE ID OF PARTS USED IN REPORT

FILE STORAGE AREA.

STORE QUANTITY OF PARTS USED IN

REPORT FILE STORAGE AREA.

COMPUTE COST OF PARTS USED.

CONVERT RESULT TO DECIMAL

STORE IN REPORT FILE AREA.

ALTER SEQUENCE: READ NEXT USAGE
RECORD AFTER WRITING CURRENT
MASTER RECORD.

0141	00212	026013R	JMP	READU	
0142	00213	000000	CONVM	NOP	
0143	00214	016006X	DLD	PRISM	STORE ID FIELDS IN COMMON
	00215	000000B			
0144	00216	016007X	DST	DT0BI	LOCATIONS TO BE PROCESSED BY
	00217	000000C			
0145	00220	016002X	JSB	BCONV	CONVERSION SUBPROGRAM. ON
0146	00221	062002C	LDA	DT0BO	COMPLETION, STORE RESULTS IN
0147	00222	070025B	STA	MTEMP	LOCATIONS USED BY PROCESSING
0148	00223	126213R	JMP	CONVM,I	SECTIONS. CONVM APPLIES TO ID OF
0149	00224	000000	CONUI	NOP	MASTER PARTS FILE; CONUI, TO ID
0150	00225	016006X	DLD	PRTSU	OF USAGE; CONU2, TO QUANTITY OF
	00226	000004B			
0151	00227	016007X	DST	DT0BI	USAGE; AND CONVC, TO COST OF
	00230	000000C			
0152	00231	016002X	JSB	BCONV	PARTS (THIS IS A BINARY TO
0153	00232	062002C	LDA	DT0BO	DECIMAL CONVERSION).
0154	00233	070026B	STA	UTEMP	
0155	00234	126224R	JMP	CONUI,I	
0156	00235	000000	CONU2	NOP	
0157	00236	016006X	DLD	PRTSU+2	
	00237	000006B			
0158	00240	016007X	DST	DT0BI	
	00241	000000C			
0159	00242	016002X	JSB	BCONV	
0160	00243	062002C	LDA	DT0BO	
0161	00244	070027B	STA	UTEMP+1	
0162	00245	126235R	JMP	CONU2,I	
0163	00246	000000	CONVC	NOP	
0164	00247	016006X	DLD	SWTMP	
	00250	000027B			
0165	00251	016007X	DST	BT0DI	
	00252	000003C			
0166	00253	016003X	JSB	DCONV	
0167	00254	016006X	DLD	BT0DO	
	00255	000005C			
0168	00256	016007X	DST	SWTMP	
	00257	000027B			
0169	00260	126246R	JMP	CONVC,I	
0170	00261	016001X	WRITC	.IOC.	WRITE ONE RECORD OF PARTS COST
0171	00262	020102	OCT	20102	REPORT ON STANDARD UNIT 2
0172	00263	026276R	JMP	RJCTC	(TELEPRINTER OUTPUT). PRISC IS
0173	00264	000010B	DEF	PRISC	ADDRESS IN STORAGE AREA; AREA IS
0174	00265	000013	DEC	11	11 WORDS LONG. RECORD IS IN ASCI
0175	00266	016001X	CKSTC	.IOC.	CHECK STATUS OF UNIT 2.
0176	00267	040002	OCT	40002	
0177	00270	002020	SSA		
0178	00271	026266R	JMP	CKSTC	IF BUSY, LOOP UNTIL FREE.
0179	00272	001200	RAL		
0180	00273	002020	SSA		
0181	00274	026004X	JMP	ABORT	TERMINATE IF ANY I/O ERROR.
0182	00275	026301R	JMP	WRITN	IF COMPLETE, TRANSFER TO WRITN.
0183	00276	006020	RJCTC	SSB	IF BUSY, LOOP UNTIL FREE.
0184	00277	026261R	JMP	WRITC	TERMINATE ON ANY OTHER REJECT
0185	00300	026004X	JMP	ABORT	CONDITION.
0186	00301	016001X	WRITN	.IOC.	WRITE ONE RECORD (BINARY) OF
0187	00302	020104	OCT	20104	NEW MASTER PARTS LIST ON UNIT 4
0188	00303	026316R	JMP	RJCTN	(TAPE PUNCH). PRISM (INPUT AREA)

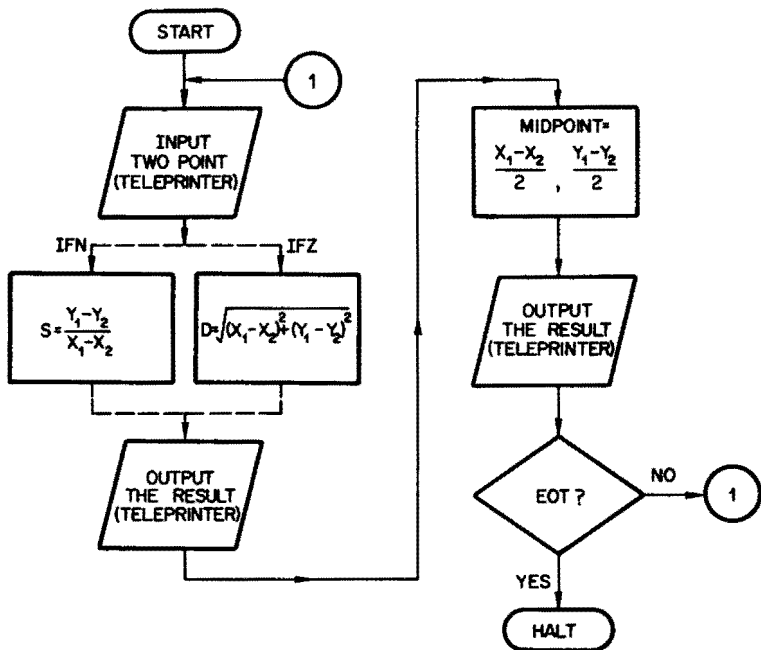
0189	00304	000000B	DEF	PRTSM	IS ALSO USED AS OUTPUT AREA.
0190	00305	000004	DEC	4	
0191	00306	016001X	CKSTN	JSB .IOC.	CHECK STATUS OF UNIT 4.
0192	00307	040004	OCT	40004	
0193	00310	002020	SSA		
0194	00311	026306R	JMP	CKSTN	IF BUSY, LOOP UNTIL FREE.
0195	00312	001200	RAL		
0196	00313	002020	SSA		
0197	00314	026004X	JMP	ABORT	
0198	00315	026013R	JMP	READU	
0199	00316	006020	RJCTN	SSB	IF BUSY, LOOP UNTIL FREE, OTHER-
0200	00317	026301R	JMP	WRITN	WISE TERMINATE.
0201	00320	026004X	JMP	ABORT	
0202			END	START	
** NO ERRORS*					

B.

Program "Line" will either calculate the distance between two points or find the slope of the line connecting the points; then the point equidistant from each point (the mid-point) is calculated.

Data is input using the formatter library routine four n-digit real numbers at a time. The first quantity is the X coordinate of the first point; the second quantity is the Y coordinate of the first point; the third and fourth quantities are the X and Y coordinates of the second point.

The result is output to the teleprinter by the formatter library routine; each quantity cannot be more than an eight digit real number.



GENERAL FLOW CHART

Below is the source program as it is typed up on the teleprinter. After it are the assembler listings. The first listing results from including the Z option in the control statement. In the second listing the Noption has been included in the control statement.

NOTE: When the complete data tape has been read and the tape reader encounters 10 blank feed frames, an EQT message is typed on the teleprinter and the computer halts. Thus no halt instruction is needed in the program.)

```

      HED LINE FORMULI:  DISTANCE, SLOPE, MID-POINT
*   PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
*   TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
*   THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
*   POINT (THE MID-POINT) IS CALCULATED.
*   DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
*   FOUR N-DIGIT REAL NUMBERS AT A TIME.  THE FIRST
*   QUANTITY IS THE X COORDINATE OF THE FIRST POINT; THE
*   SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
*   THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
*   OF THE SECOND POINT.
*   THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
*   FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
*   THAN AN EIGHT DIGIT REAL NUMBER.
      NAM LINE
START NOP
      JMP INPUT
      EXT .IOC.,FLOAT,IFIX,SQRT
      EXT .DIO.,.IOI.,.DTA.,.RAR.
      EXT .IOR.,.IAR.
.DATA DEF DATA
.PRIN DEF PRINT
DATA BSS 4
FMT  ASC 3,(F8.3)
FMT2  ASC 8,(F8.3,"",F8.3/)
FMT3  ASC 3,(4I2)
      SKP
*   INPUT THE FIRST TWO POINTS; FOUR DATA WORDS
INPUT NOP
      LDA =B5
      CLB,INB
      JSB .DIO.
      DEF FMT3
      DEF **4
      LDA =B4
      LDB .DATA
      JSB .IAR.
      SPC 3
*   THE DISTANCE BETWEEN THE TWO POINTS:
      IFZ
      LDA DATA+2
      CMA,INA
      ADA DATA
      SPC 1
      JMP **5
PRINT REP 4
      NDP
      SPC 1
      STA PRINT
      SUP

```

```

    MPY PRINT
    STA PRINT
    SPC 1
    LDA DATA+3
    CMA, INA
    ADA DATA+1
    STA PRINT+1
    MPY PRINT+1
    ADA PRINT
    SPC 1
    JSB FLOAT
    JSB SQRT
    DST PRINT
    XIF
    SPC 3
*   FIND THE SLOPE OF THE LINE
    IFN
    LDA DATA+2
    CMA, INA
    ADA DATA
    JMP **5
PRINT REP 4
    NOP
    STA PRINT
    SPC 1
    LDA DATA+3
    CMA, INA
    ADA DATA+1
    CLB
    DIV PRINT
    DST PRINT
    XIF
    SPC 3
*   OUTPUT THE RESULT
    LDA =B2
    CLB
    JSB .D10.
    DEF FMT
    DEF **4
    OLD PRINT
    JSB .I0R.
    JSB .DTA.
    SPC 3
*   FIND THE MID-POINT OF THE LINE SEGMENT:
    LDA DATA
    ADA DATA+2
    CLB
    JSB FLOAT
    FMP =F.5
    DST PRINT
    SPC 1
    LDA DATA+1
    ADA DATA+3
    CLB
    JSB FLOAT
    FMP =F.5
    DST PRINT+2
    SPC 1
    UNL

```

```

LDA =B2
CLB
JSB .DIO.
DEF FMT2
DEF **5
LDA =B2
LDB .PRIN
JSB .RAR.
JSB .DTA.
LST
SPC 3
UNS
JMP INPUT
END START

```

0001

ASMB,R,L,T,Z

START R 000000
.IOC. X 000001
FLOAT X 000002
IFIX X 000003
SQRT X 000004
.DIO. X 000005
.IOI. X 000006
.DTA. X 000007
.RAR. X 000010
.IOR. X 000011
.IAR. X 000012
.DATA R 000002
.PRIN R 000003
DATA R 000004
FMT R 000010
FMT2 R 000013
FMT3 R 000023
INPUT R 000026
PRINT R 000043
.MPY X 000013
.DST X 000014
.DLD X 000015
.FMP X 000016
** NO ERRORS*

```

0002* PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
0003* TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
0004* THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
0005* POINT (THE MID-POINT) IS CALCULATED.
0006* DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
0007* FOUR N-DIGIT REAL NUMBERS AT A TIME. THE FIRST
0008* QUANTITY IS THE X COORDINATE OF THE FIRST POINT;THE
0009* SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
0010* THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
0011* OF THE SECOND POINT.
0012* THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
0013* FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
0014* THAN AN EIGHT DIGIT REAL NUMBER.
0015 00000 NAM LINE
0016 00000 000000 START NOP
0017 00001 026026R JMP INPUT
0018 EXT .10C.,FLOAT,IFIX,SQRT
0019 EXT .D10.,.101.,.DTA.,.RAR.
0020 EXT .10R.,.1AR.
0021 00002 000004R .DATA DEF DATA
0022 00003 000043R .PRIN DEF PRINT
0023 00004 000000 DATA BSS 4
0024 00010 024106 FMT ASC 3,(F8.3)
00011 034056
00012 031451
0025 00013 024106 FMT2 ASC 8,(F8.3,"",F8.3/)
00014 034056
00015 031454
00016 021054
00017 021054
00020 043070
00021 027063
00022 027451
0026 00023 024064 FMT3 ASC 3,(412)
00024 044462
00025 024440

```

0028* INPUT THE FIRST TWO POINTS; FOUR DATA WORDS

```

0029 00026 000000 INPUT NOP
0030 00027 062131R LDA =B5
0031 00030 006404 CLB,INB
0032 00031 016005X JSB .D10.
0033 00032 000023R DEF FMT3
0034 00033 000037R DEF **4
0035 00034 062132R LDA =B4
0036 00035 066002R LDB .DATA
0037 00036 016012X JSB .IAR.

```

0039* THE DISTANCE BETWEEN THE TWO POINTS:

```

0040 IFZ
0041 00037 062006R LDA DATA+2
0042 00040 003004 CMA,INA
0043 00041 042004R ADA DATA

0045 00042 026047R JMP **5
0046 PRINT REP 4
0047 00043 000000 NOP
0047 00044 000000 NOP
0047 00045 000000 NOP
0047 00046 000000 NOP

0049 00047 072043R STA PRINT
0050 SUP
0051 00050 016013X MPY PRINT
0052 00052 072043R STA PRINT

0054 00053 062007R LDA DATA+3
0055 00054 003004 CMA,INA
0056 00055 042005R ADA DATA+1
0057 00056 072044R STA PRINT+1
0058 00057 016013X MPY PRINT+1
0059 00061 042043R ADA PRINT

0061 00062 016002X JSB FLOAT
0062 00063 016004X JSB SQRT
0063 00064 016014X DST PRINT
0064 XIF

```

0066* FIND THE SLOPE OF THE LINE

```

0067 IFN
0068 LDA DATA+2
0069 CMA,INA
0070 ADA DATA
0071 JMP **5
0072 PRINT REP 4
0073 NOP
0074 STA PRINT
0075 SPC 1
0076 LDA DATA+3
0077 CMA,INA
0078 ADA DATA+1

```



```
0079          CLB
0080          DIV PRINT
0081          DST PRINT
0082          XIF
```

0084* OUTPUT THE RESULT

```
0085 00066 062133R LDA =B2
0086 00067 006400 CLB
0087 00070 016005X JSB .D10.
0088 00071 000010R DEF FMT
0089 00072 000076R DEF **4
0090 00073 016015X DLD PRINT
0091 00075 016011X JSB .10R.
0092 00076 016007X JSB .DTA.
```

0094* FIND THE MID-POINT OF THE LINE SEGMENT:

```
0095 00077 062004R LDA DATA
0096 00100 042006R ADA DATA+2
0097 00101 006400 CLB
0098 00102 016002X JSB FLOAT
0099 00103 016016X FMP =F.5
0100 00105 016014X DST PRINT

0102 00107 062005R LDA DATA+1
0103 00110 042007R ADA DATA+3
0104 00111 006400 CLB
0105 00112 016002X JSB FLOAT
0106 00113 016016X FMP =F.5
0107 00115 016014X DST PRINT+2
```

```
0119          LST
```

```
0121          UNS
0122 00130 026026R JMP INPUT
      00131 000005
      00132 000004
      00133 000002
      00134 040000
      00135 000000
```

```
0123          END START
```

** NO ERRORS*

PAGE 0001

0001

ASMB,R,L,T,N

START R 000000
.IOC. X 000001
FLOAT X 000002
IFIX X 000003
SQRT X 000004
.DIO. X 000005
.IOI. X 000006
.DTA. X 000007
.RAR. X 000010
.IOR. X 000011
.IAR. X 000012
.DATA R 000002
.PRIN R 000003
DATA R 000004
FMT R 000010
FMT2 R 000013
FMT3 R 000023
INPUT R 000026
PRINT R 000043
.DIV X 000013
.DST X 000014
.DLD X 000015
.FMP X 000016
** NO ERRORS*

```

0002* PROGRAM LINE WILL EITHER CALCULATE THE DISTANCE BETWEEN
0003* TWO POINTS OR FIND THE SLOPE OF THE LINE CONNECTING
0004* THE POINTS; THEN THE POINT EQUIDISTANT FROM EACH
0005* POINT (THE MID-POINT) IS CALCULATED.
0006* DATA IS INPUT USING THE FORMATTER LIBRARY ROUTINE
0007* FOUR N-DIGIT REAL NUMBERS AT A TIME. THE FIRST
0008* QUANTITY IS THE X COORDINATE OF THE FIRST POINT;THE
0009* SECOND QUANTITY IS THE Y COORDINATE OF THE FIRST POINT;
0010* THE THIRD AND FOURTH QUANTITIES ARE THE X AND Y COORDINATES
0011* OF THE SECOND POINT.
0012* THE RESULT IS OUTPUT TO THE TELEPRINTER BY THE
0013* FORMATTER LIBRARY ROUTINE; EACH QUANTITY CANNOT BE MORE
0014* THAN AN EIGHT DIGIT REAL NUMBER.
0015 00000 NAM LINE
0016 00000 000000 START NOP
0017 00001 026026R JMP INPUT
0018 EXT .IOC.,FLOAT,IFIX,SQRT
0019 EXT .DIO.,.IOI.,.DTA.,.RAR.
0020 EXT .IOR.,.IAR.
0021 00002 000004R .DATA DEF DATA
0022 00003 000043R .PRIN DEF PRINT
0023 00004 000000 DATA BSS 4
0024 00010 024106 FMT ASC 3,(F8.3)
      00011 034056
      00012 031451
0025 00013 024106 FMT2 ASC 8,(F8.3,"",F8.3/)
      00014 034056
      00015 031454
      00016 021054
      00017 021054
      00020 043070
      00021 027063
      00022 027451
0026 00023 024064 FMT3 ASC 3,(412)
      00024 044462
      00025 024440

```

0028* INPUT THE FIRST TWO POINTS; FOUR DATA WORDS

```

0029 00026 000000 INPUT NOP
0030 00027 062123R LDA =B5
0031 00030 006404 CLB,INB
0032 00031 016005X JSB .D10.
0033 00032 000023R DEF FMT3
0034 00033 000037R DEF **4
0035 00034 062124R LDA =B4
0036 00035 066002R LDB .DATA
0037 00036 016012X JSB .IAR.

```

0039* THE DISTANCE BETWEEN THE TWO POINTS:

```

0040 IFZ
0041 LDA DATA+2
0042 CMA,INA
0043 ADA DATA
0044 SPC 1
0045 JMP **5
0046 PRINT REP 4
0047 NOP
0048 SPC 1
0049 STA PRINT
0050 SUP
0051 MPY PRINT
0052 STA PRINT
0053 SPC 1
0054 LDA DATA+3
0055 CMA,INA
0056 ADA DATA+1
0057 STA PRINT+1
0058 MPY PRINT+1
0059 ADA PRINT
0060 SPC 1
0061 JSB FLOAT
0062 JSB SQRT
0063 DST PRINT
0064 XIF

```

0066* FIND THE SLOPE OF THE LINE

```

0067 IFN
0068 00037 062006R LDA DATA+2
0069 00040 003004R CMA,INA
0070 00041 042004R ADA DATA
0071 00042 026047R JMP **5
0072 PRINT REP 4
0073 00043 000000 NOP
0073 00044 000000 NOP
0073 00045 000000 NOP
0073 00046 000000 NOP
0074 00047 072043R STA PRINT

0076 00050 062007R LDA DATA+3
0077 00051 003004R CMA,INA
0078 00052 042005R ADA DATA+1

```

```

0028* INPUT THE FIRST TWO POINTS; FOUR DATA WORDS
0029 00026 000000 INPUT NOP
0030 00027 062123R LDA =B5
0031 00030 006404 CLB,INB
0032 00031 016005X JSB .DIO.
0033 00032 000023R DEF FMT3
0034 00033 000037R DEF **4
0035 00034 062124R LDA =B4
0036 00035 066002R LDB .DATA
0037 00036 016012X JSB .IAR.

```

```

0039* THE DISTANCE BETWEEN THE TWO POINTS:

```

```

0040 IFZ
0041 LDA DATA+2
0042 CMA,INA
0043 ADA DATA
0044 SPC 1
0045 JMP **5
0046 PRINT REP 4
0047 NOP
0048 SPC 1
0049 STA PRINT
0050 SUP
0051 MPY PRINT
0052 STA PRINT
0053 SPC 1
0054 LDA DATA+3
0055 CMA,INA
0056 ADA DATA+1
0057 STA PRINT+1
0058 MPY PRINT+1
0059 ADA PRINT
0060 SPC 1
0061 JSB FLOAT
0062 JSB SQRT
0063 DST PRINT
0064 XIF

```

```

0066* FIND THE SLOPE OF THE LINE

```

```

0067 IFN
0068 00037 062006R LDA DATA+2
0069 00040 003004 CMA,INA
0070 00041 042004R ADA DATA
0071 00042 026047R JMP **5
0072 PRINT REP 4
0073 00043 000000 NOP
0073 00044 000000 NOP
0073 00045 000000 NOP
0073 00046 000000 NOP
0074 00047 072043R STA PRINT

0076 00050 062007R LDA DATA+3
0077 00051 003004 CMA,INA
0078 00052 042005R ADA DATA+1

```

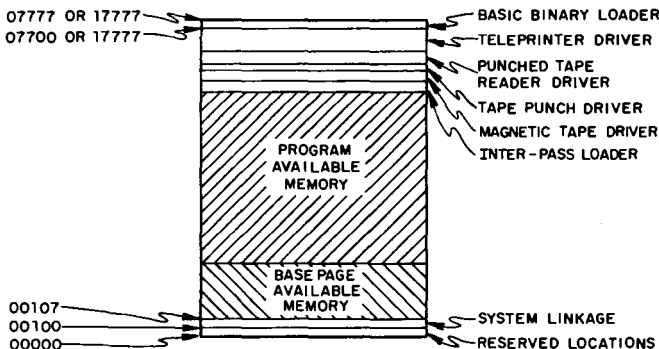
The System Input/Output (SIO) subroutines may be used to perform basic input/output operations for programs in absolute form. † Subroutines are available for:

2752A Teleprinter Driver
2737A Punched Tape Reader Driver
2753A Tape Punch Driver
2020A Magnetic Tape Unit Driver
2020A Magnetic Tape Unit Inter-Pass Loader

MEMORY ALLOCATION

These drivers are stored in high memory immediately preceding the Basic Binary Loader. The Teleprinter driver must be loaded first; it is stored in the highest portion of this area. The drivers for the Punched Tape Reader, the Tape Punch and the Magnetic Tape Unit may then be loaded; sequence of loading does not effect operation. However, the drivers will be stored in memory below the Teleprinter Driver in the following order: Punched Tape Reader Driver, Tape Punch Driver, Magnetic Tape Driver, and Inter-Pass Loader if needed.

The drivers are accessed through 15-bit absolute addresses which are stored in the System Linkage area starting at location 101₈. The allocation of memory is as follows:



† The SIO subroutines are designed for use with the standard software systems, FORTRAN, Assembler, and Symbolic Editor, however, they may be used with any absolute object program.

OPERATION AND CALLING SEQUENCE:

PAPER TAPE DEVICES

All data transmission is accomplished without interrupt control, and therefore, operations are not buffered by the drivers. Control is not returned to the calling program until an operation is completed. Data is transferred to and from buffer storage areas specified in the user program.

The general form of the paper tape input/output calling sequence is:

LDA <buffer length> (words or characters)

LDB <buffer address>

JSB 10fB,I (f is Input/Output function)

<normal return>

Register Contents

When the JSB is performed, the A-Register must contain the length of the buffer storage area and the B-Register, the address of the buffer. Control returns to the location following the JSB. After an input request is completed, the A-Register contains a positive integer indicating the number of characters or words transmitted, or zeros, if an end-of-tape condition occurred.

The digit supplied for f in the JSB instruction determines the paper tape input/output function to be performed. The value of the operand address is the location in the System Linkage that contains the absolute address of the driver entry point. The following are available:

- 101 Input
- 102 List Output
- 103 Punch Output
- 104 Keyboard Input—ASCII data is read from Teleprinter and printed as it is received.

If the Teleprinter driver alone is loaded, these locations point to entry points of this driver. If Punched Tape Reader and Tape Punch drivers are in memory, location 101 points to the Punched Tape Reader driver and location 103, to the Tape Punch driver. If the latter are to be used, they must be loaded after the Teleprinter driver.

OPERATION AND CALLING SEQUENCE:

MAGNETIC TAPE DRIVER

As with the Paper Tape SIO drivers, all data transmission is accomplished without interrupt control. Control is not returned to the calling program until an operation is completed. (Rewind and rewind standby are the only exceptions to this. In these cases return is made as soon as the command is accepted.)

The general form of the calling sequence is:

```
LDA <buffer length> or <file count>
LDB <buffer address> or <record count>
JSB 107B,I
OCT <command code>
<EOF/EOT return>
<error return>
<normal return>
```

NOTE: Location 107_B must contain the address of the magnetic tape driver.

Register Contents

Before initiating read or write operations, the A-Register must contain the buffer length. This will be a positive integer if length is defined in characters and a negative integer if length is defined in words. The B-Register must contain the buffer address.

Before initiating tape positioning operations, the A-Register must contain the number of files that are to be spaced. A positive integer indicates forward spacing; a negative integer indicates backward spacing. The B-Register contains the number of records that are to be spaced. A positive integer indicates forward spacing; a negative integer indicates backward spacing. The positioning may be defined in terms of any combination of forward or backward spacing of files and records (e.g., space forward two files then backspace three records). If files only or records only are to be spaced, the contents of the other register should be zeros.

The registers are not used when entering the subroutine to perform one of the following operations:

Write end-of-file
Write file gap
Rewind

Rewind and Standby
Status

Linkage Address

107B is the System Linkage word that contains the absolute address of the entry point for the Magnetic Tape driver.

On return from a read or write operation, the A-Register contains a positive value indicating the number of words or characters transmitted.

On return from all operations except Rewind and Rewind and Standby the B-Register contains status of the operation (See Status).

MAGNETIC TAPE OPERATIONS

The magnetic tape driver will perform the following operations. The pertinent operation is specified by the command code which appears after the OCT in the calling sequence.

<u>Operation</u>	<u>Command Code</u>
Read	0
Write	1
Write End-of-File	2
Rewind	3
Position	4
Rewind and Standby	5
Gap	6
Status	7

Read

One tape record is read into the buffer. The number of characters or words read is stored in the A-Register. The value will be equal to the buffer length except when the data on tape is less than the length of the buffer. If this occurs, the value in the A-Register will be the amount of data recorded on tape. If the tape record exceeds the buffer length, the data will be read into the buffer until the buffer is filled, the remainder of the record will be skipped. If the length of an input buffer is an odd number of characters, a read operation will result in the overlaying of the character following the last character of the buffer; the subroutine actually transmits full words only.

If end-of-tape exists on entry or an end-of-file character is encountered during the operation, control returns to the error return address. Five attempts are made to read the record before returning control to the error return address.

Write

The contents of the buffer is written on tape preceded by the record length. Since a minimum of 7 tape characters may be written, short records are padded by the subroutine.

If the end-of-tape is detected during the write operation, the normal return is used. The next write operation, however, results in a return of control of the EOF/EOT location; no data is written.

If an error is encountered during a write operation, the tape is backspaced, 3 inches are erased, and another write is attempted. The driver returns to the error return location if the error persists for five attempts.

Write End-of-File

A standard EOF character (17g) is written on tape. Control returns to the normal location with the EOF status on the B-Register. No gap is written.

If the end of tape was reached on a previous write command, control returns to the EOF/EOT location; the character is written.

Rewind

This command initiates a rewind operation and then immediately returns control to the normal location.

The calling sequence for a Rewind operation consists of:

```
JSB      107B,I  
OCT      3  
<normal return>
```

The user need not test status on the rewind operation before issuing the next call.

Position

This is the general command to move the tape. Both record and file operations may be defined in the same operation. Either may be specified for forward or backward spacing. At the completion of the operation the tape will be positioned ready for reading or writing.

An attempt to space beyond the End-of-Tape or Start-of-Tape will terminate the positioning operation and return control to the EOF/EOT location.

Rewind and Standby

This causes the tape to be positioned at load point and switches the device to local status. If the tape is at load point when the command is given, it is interpreted as a rewind command only; the device is not switched to local status. Control returns to the normal location immediately after the operation is initiated.

The calling sequence for a Rewind and Standby operation consists of:

```
JSB      107B,I  
OCT      5  
<normal return>
```

An attempt to issue another call on this device results in a halt (102044). The device must be switched to AUTO before the program can continue.

Gap

This command causes a 3-inch gap to be written on the tape.

If the End-of-Tape was reached on a previous write command, control returns to the EOF/EOT location; the gap is not written.

Status

This command returns certain status bits in the B-Register. The driver performs a clear command whenever it is entered and as a result the only bits that are valid indicators are:

Start-of-Tape
End-of-Tape
Write Not Enabled

All other commands (except Rewind and Rewind and Standby) provide valid status replies on return to the program.

The status reply consists of bits 8-0 and has the following significance:

<u>Bits 8-0</u>	<u>Condition</u>
1xxxxxxx	Local - The device is in local status
x1xxxxxxx	EOF- An End-of-File character (17g) has been detected while reading, forward spacing, or backspacing.
xx1xxxxxx	SOT - The Start-of-Tape marker is under the photo sense head.
xxx1xxxxx	EOT - The End-of-Tape reflective marker is sensed while the tape is moving forward. The bit remains set until a rewind command is given.
xxxx1xxxx	Timing - The data channel flag has not been cleared.
xxxxx1xxx	Reject - a) Tape motion is required and the unit is busy. b) Backward tape motion is required and the tape is at load point. c) A write command is given and the tape reel does not have a write enable ring.
xxxxxx1xx	Write not enabled - Tape reel does not have write enable ring or tape unit is rewinding.
xxxxxxx1x	Parity error - A vertical or longitudinal parity error occurred during reading or writing. (Parity is not checked during forward or backward spacing operations.)
xxxxxxx1	Busy - The tape is in motion or the device is in local status.

Following is a table summarizing the tape commands:

	Command Code	Call		Return	
		A	B	A	B
Read	0	Buffer Length	Buffer Address	Buffer or Record Length	Status
Write	1	Buffer Length	Buffer Address	Buffer Length	Status
Write EOF	2	-	-	-	Status
Rewind	3	-	-	-	-
Position	4	Number of Files, Direction	Number of Records, Direction	-	Status
Rewind and Standby	5	-	-	-	-
Gap	6	-	-	-	Status
Status	7	-	-	-	Partial Status

Error Messages

Tape Unit in Local Status:

The subroutine halts with 102044 in the T-Register. Switch tape unit to AUTO mode and press RUN to continue.

Write Not Enabled:

The subroutine halts with 102011 in the T-Register. The error is irrecoverable.

Additional Linkage Addresses

Other locations in the System Linkage area contain the following:

100_h Used by the standard software system to store a JMP to the transfer address.

105_h First word address of available memory.

106_h Last word address of available memory.

The latter two locations may be accessed by an absolute program. The user may store the first word of available memory in 105 by performing the following:

ORG 105B

ABS < last location of user program +1 >

The last word of available memory is established by the drivers; it is the location immediately preceding the first location used by the last driver loaded.

BUFFER STORAGE AREA

The Buffer Address is the location of the first word of data to be written on an output device or the first word of a block reserved for storage of data read from an input device. The length of the buffer area is specified in the A-Register in terms of ASCII input or output characters or binary output words. For binary input, the length of the buffer is the length of the record which is specified in the first character of the record. ASCII and binary input record lengths are given as positive integers. The length of a binary output record is specified as the two's complement of the number of words in the record.

In addition to describing the buffer area in the calling sequence, (or first word of binary input record), the area must also be specifically defined in the program, for example with a BSS instruction.

Record Formats

ASCII Records (Paper Tape)

An ASCII record is a group of characters terminated by an end-of-record mark which consists of a carriage return, (CR), and a line feed, (LF).

For an input operation, the length of the record transmitted to the buffer is the number of characters designated in the A-Register, or less if an end-of-record mark is encountered before the character count is exhausted. The codes for (CR) and (LF) are not transmitted to the buffer. An end-of-record mark preceding the first data character is ignored.

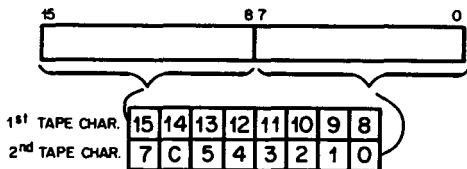
For an output operation, the length of the record is determined by the number of characters designated in the request. An end-of-record mark is supplied at the end of each output operation by the driver.

If a (RUB OUT) code followed by a (CR) (LF) is encountered on input from the Teleprinter or Punched Tape Reader, the current record is ignored (deleted) and the next record transmitted. †

If less than ten feed frames (all zeros) are encountered before the first data character from the Punched Tape Reader, they are ignored. Ten feed frames are interpreted as an end-of-tape condition.

Binary Records (Paper Tape)

A binary record is transmitted exactly as it appears in memory or on 8-level paper tape. Each computer word is translated into two tape "characters" (and vice versa) as follows:



For an output operation, the record length is the number of words designated by the value in the A-Register (the value is the two's complement of the number of words). For input operations, the first word of the record contains a positive integer in bits 15-8 specifying the length (in words) of the record including the first word.

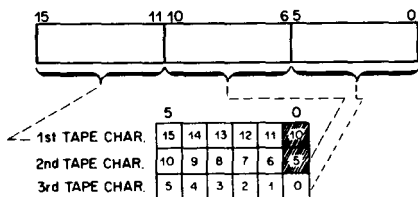
† (RUB OUT) which appears on the Teleprinter keyboard is synonymous with the ASCII symbol (DEL)

On input operations if less than ten feed frames precede the first data character, they are ignored; ten feedframes are interpreted as an end-of-tape condition. On output, the driver writes four feed frames to serve as a physical record separator.

Binary Records (Magnetic Tape)

The Magnetic Tape subroutine reads and writes binary (odd parity) records only. A record count is supplied by the driver as the first word of the record. This allows automatic padding of short records to the minimum record length with automatic removal of the padded portion of the record on read.

Each Computer word is translated into three tape "characters" (and vice versa) as follows:



INTER-PASS LOADER

The Inter-Pass Loader is a System Input/Output (SIO) subroutine most commonly used to locate and load the second pass of FORTRAN. It may also be used by any absolute program to locate program segments on magnetic tape and load them in memory overlaying program segments that have already been executed.

The request to read in a new segment is initiated by performing a JMP instruction with the A-Register containing the identification number of the program segment to be loaded. (The number is assigned to the program segment during Prepare Tape System processing.) When the new program has been located and loaded, the Inter-Pass Loader passes control to it via a JMP 3,I.

The calling sequence of the program segment initially in memory is:

```
LDA    <identification>
JMP    6,I
```

where identification is an octal number in the range 1-77776.

The first instructions of the program to be loaded by the Inter-Pass Loader might be:

```
ORG    3
DEF    start of segment
```

CALLING SEQUENCES

The Formatter is a library subroutine used by FORTRAN and ALGOL to input or output data. An assembler program may access the Formatter routine with a 5 to 9 line calling sequence depending on the form of the call.

I. Format Definition

	<u>INPUT</u>	<u>OUTPUT</u>
Formatted	LDA (unit)	LDA (unit)
	CLB, INB	CLB
	JSB .DIO.	JSB .DIO.
	DEF (fmt) or ABS 0	DEF (fmt)
	DEF (end of list)	DEF (end of list)
Binary	LDA (unit)	LDA (unit)
	CLB, INB	CLB
	JSB .BIO.	JSB .BIO.

where

unit	refers to the unit reference number of the device to be called
fmt	is the label of an ASC pseudo instruction which defines the format specification
end of list	is the location immediately following the last parameter of the calling sequence; it is to this location that the Formatter returns control.
ABS 0	is an option for free field input
formatted input/output	is in ASCII code
binary input/output	is in binary code

II. Element Definition

	<u>INPUT</u>	<u>OUTPUT</u>
Real Variable	JSB .IOR. DST x	DLD x JSB .IOR.
Integer Variable	JSB .IOI, STA i	LDA i JSB .IOI.
Array	LDA array length LDB array address JSB .RAR. (real) or .IAR. (integer)	

where

x or i are addresses, real or integer, of the data
array length is the number of elements (not the number of memory locations) in the block of data.
(Maximum length is equivalent to 60 computer words.)

III. Terminator

<u>INPUT</u>	<u>OUTPUT</u>
(none)	JSB .DTA.

Symbols such as .DIO., .IOR., etc., are entry points to the Formatter; all entry points used in the calling sequence must be declared external with an EXT pseudo code.

Data stored in memory may be converted internally from one format to another with the following initial call.

```
LDA    =BO
JSB    .DIO.
DEF    buffer
DEF    (fmt)
DEF    (end of list)
```

·
·

Element Definition

·

Terminator

where buffer is the address of the data to be converted.

FORMAT SPECIFICATIONS

Below are listed the format conversion and editing specifications.

rAw	Alphanumeric character
rEw.d	Real number with exponent
rFw.d	Real number without exponent
rIw	Decimal integer
r@w } rKw }	Octal integer
nX	Blank field descriptor
nHh ₁ . . . h _n , }	Heading and labeling descriptors
r'h ₁ . . . h _n }	
r/	Begin new record

where

r	is the number of times the entire format is repeated
w	is the number of digits in the format
d	is the number of digits to the right of the decimal point (w-d should be greater than or equal to 4)
n	is the number of characters or spaces
h's	represents the ASCII characters
Aw	translates alphanumeric data to or from memory. If w is greater than 2 only the last two characters are processed; if w is 1, the single character is read into or written from the right-half of the computer word.
Ew	converts data to a real number. On output, data may consist of integer, fraction, and exponent subfields.

$$\frac{+}{N} n \dots n . n \dots n \frac{+}{E} ee$$

On output, data appears in floating point form.

$$\wedge . x_1 \dots x_d E \pm ee$$

Fw For output operations real numbers in memory are converted to character form which will appear right justified in decimal form. Input is identical to the E specification input.

$$\triangle x \dots x . x \dots x$$

Iw translates decimal integers to or from memory

$$\triangle x_1 \dots x_d$$

@w and Kw translates octal integers to or from memory.

$$\triangle x_1 \dots x_d$$

nHh₁ . . . h_n provides for the transfer of any combination of 8-bit ASCII characters, including blanks.

r" h₁ . . . h_n " also transfers ASCII characters; field length is not specified, quotation marks are not transferred.

(For a more detailed description of the Format specifications see the FORTRAN Programmer's Reference Manual, Section 7.)

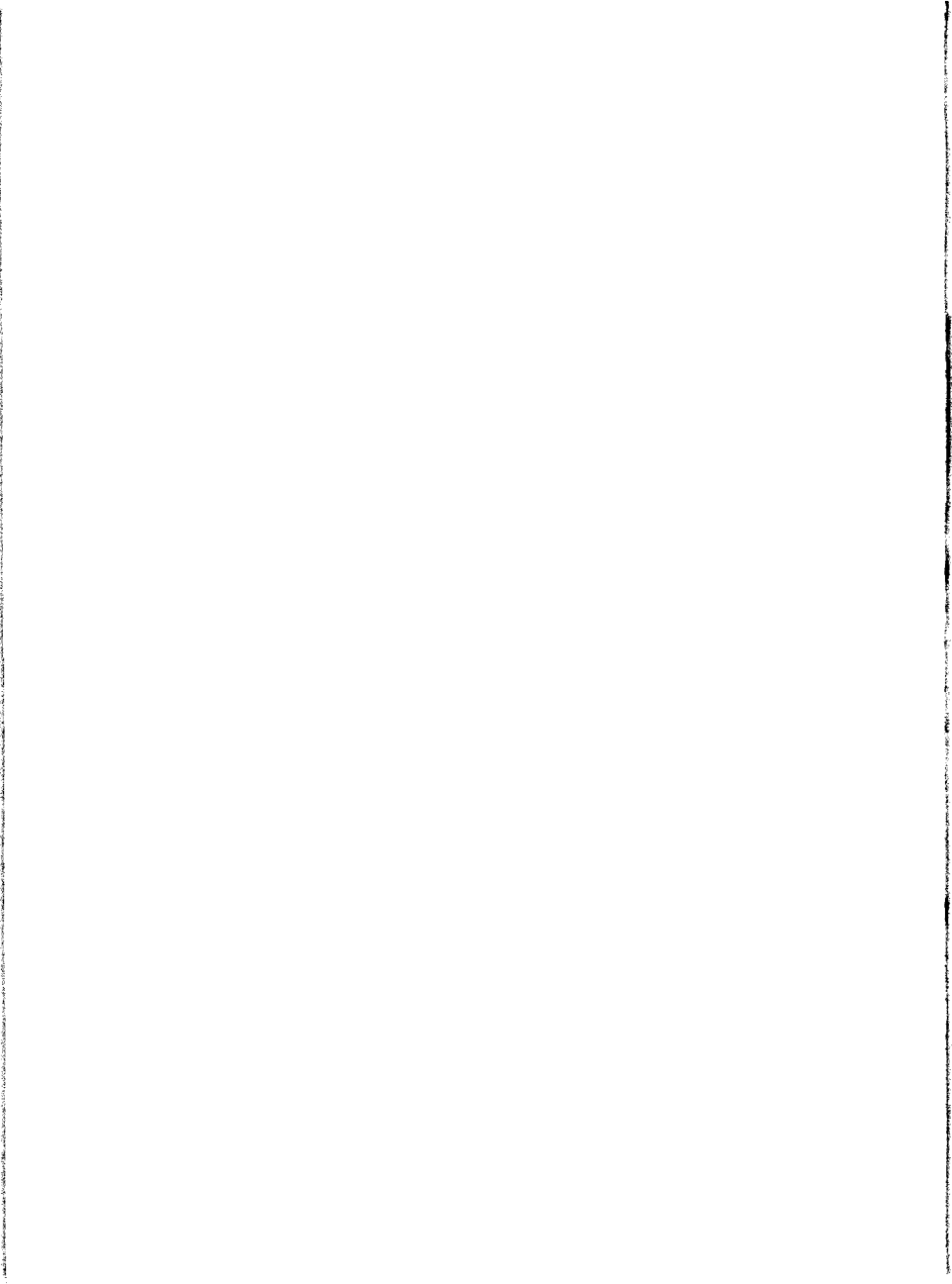
EXAMPLE

Below is an example of a calling sequence to the Formatter that will output the contents of a block data, SOLVE, such that each number is printed on the teleprinter in the following manner:

xxxxxx.xx

SOLVE occupies 100₁₀ memory locations; the data stored there is in floating point form.

1	5	10	15	20	25	30	35	40	45	50
		EXT	.DIO.	.RAR.	.DTA.					
FRMT	ASC	5,	(2X,	F8.2)						
SOLVE	BSS	100								
	*									
	*									
	*									
	LDA	=B5								
	CLB									
	JSB	.DIO.								
	DEF	FRMT								
	DEF	*+5								
	LDA	=D50								
	LDB	SOLVE								
	JSB	.RAR.								
	JSB	.DTA.								



CROSS REFERENCE TABLE GENERATOR G

The Cross Reference Table Generator routine processes an Assembler source program and prints a cross reference list of all symbols appearing in the program. The list contains the symbols in alphabetic order. Each is followed by the 4-digit sequence number of the statement in which the symbol was defined and the sequence numbers of all statements referring to the symbol. If the source program is contained on more than one tape, the tape number follows the statement sequence number. The tape number is determined by the order in which the tapes are submitted to the generator routine; it is not printed for the first tape. The general format of the list is as follows:

sssss dddd/tt rrrr/tt rrrr/tt rrrr/tt rrrr/tt rrrr/tt

sssss = symbol
dddd = defining statement number
tt = tape number
rrrr = reference statement numbers

Example:

The program;

```
(0001)      NAM TESTT
(0002)  BEGIN DLD A
(0003)      FMP A
(0004)      DST A
(0005)  TEST  ISZ I
(0006)      JMP BEGIN
(0007)      HLT 3
(0008)      COM A (2),I
(0009)      END
```

yields the cross reference table:

A	0008	0002	0003	0004
BEGIN	0002	0006		
I	0008	0005		
TEST	0005			

OPERATING PROCEDURES

- A. Set Teleprinter to LINE and check that all equipment to be used is operable.
- B. Load Cross Reference Table Generator using the Basic Binary Loader. †
 - 1. Place Cross Reference Table Generator in the unit serving as the Standard Input unit (e. g. , Punched Tape Reader).
 - 2. Set Switch Register to starting address of Basic Binary Loader (e. g. , 007700 for 4K memory, 017700 for 8K memory).
 - 3. Press LOAD ADDRESS.
 - 4. Set Loader switch to ENABLE.
 - 5. Press PRESET.
 - 6. Press RUN.
 - 7. When the computer halts and indicates that the Cross Reference Table Generator is loaded (T-Register contains 102077), set Loader Switch to PROTECTED.
- C. Set Switch Register to starting address of Cross Reference Table Generator.

000100

- D. Press LOAD ADDRESS.
- E. Place source language tape in unit serving as the Standard Input unit (e. g. , Punched Tape Reader).
- F. Press RUN.
- G. At the end of each tape other than the last, the message "END OF TAPE" is printed and the computer halts. Repeat E and F.

† The appropriate System Input/Output subroutines (drivers) are assumed to be included with the Cross Reference Table Generator program.

- H. At the end of the last tape (the tape containing the END statement), the table is printed on the Standard List Output device (e. g. , Teleprinter). When the table is printed, the computer halts.

During the operation of the routine, the following may be printed:

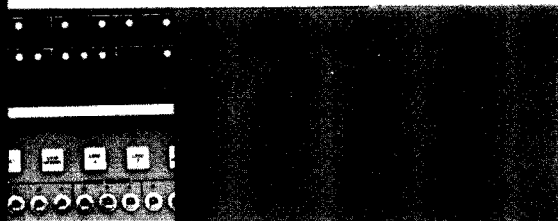
<u>Teleprinter Message</u>	<u>Explanation</u>	<u>Action</u>
DD symbol	A doubly defined symbol has been encountered. The computer does not halt.	Correct source program after completion of routine.
TABLE OVERFLOW	The combined number of symbols and references to them exceeds the capacity of the routine.	Irrecoverable error. If the Table is necessary, the source program must be revised.

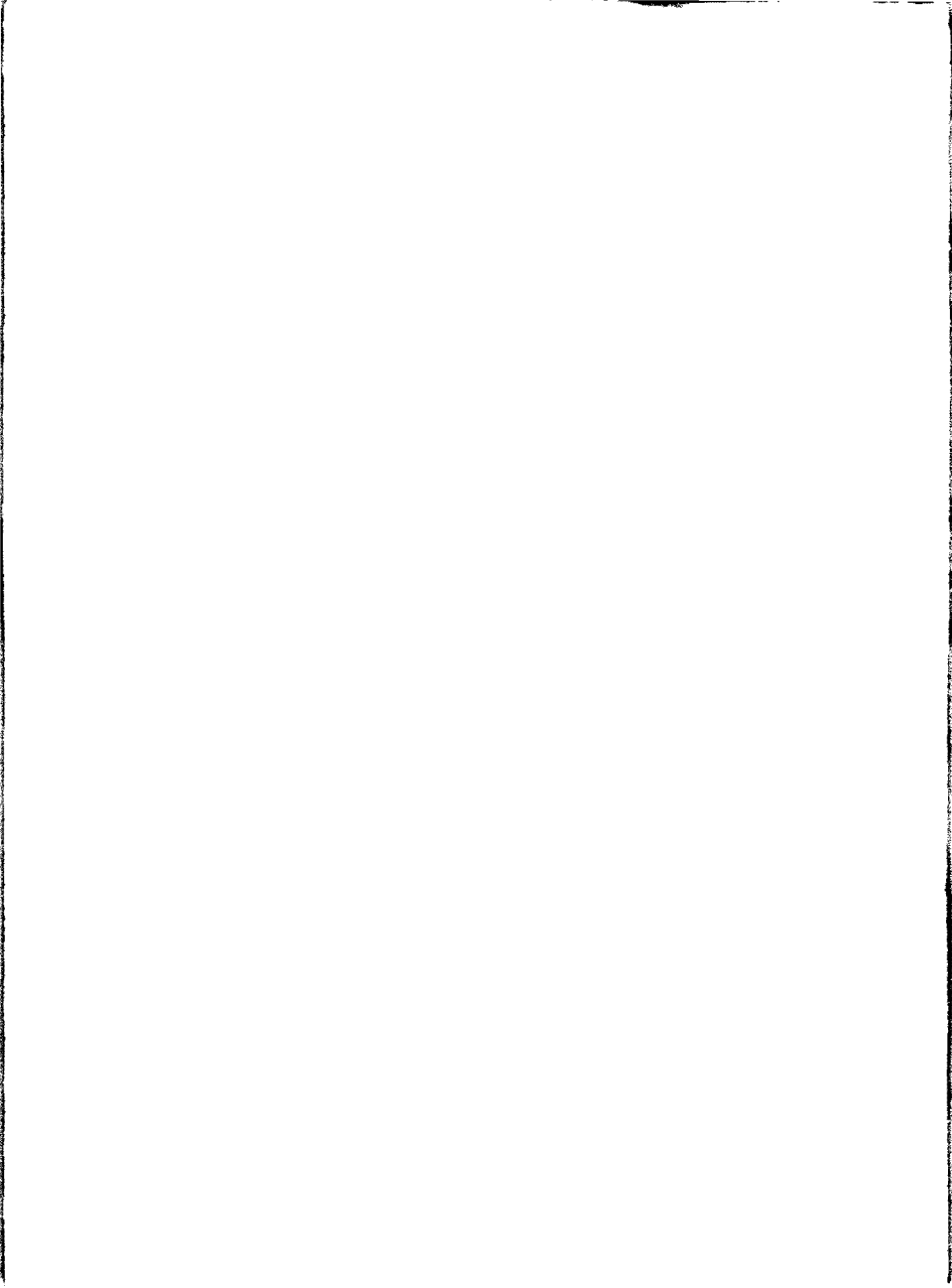
CONSOLIDATED CODING SHEET

H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
D/I	AND	001		0	Z/C		Memory Address									
D/I	XOR	010		0	Z/C											
D/I	IOR	011		0	Z/C											
D/I	JSB	001		1	Z/C											
D/I	JMP	010		1	Z/C											
D/I	ISZ	011		1	Z/C											
D/I	AD*	100		A/B	Z/C											
D/I	CP*	101		A/B	Z/C											
D/I	LD*	110		A/B	Z/C											
D/I	ST*	111		A/B	Z/C											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	SRG	000		A/B	0	D/E	*LS	000		CLE	D/E	SL*	*LS	000		
				↓	↓	↓	*RS	001			↓		*RS	001		
							R*L	010					R*L	010		
							R*R	011					R*R	011		
							*LR	100					*LR	100		
							ER*	101					ER*	101		
							EL*	110					EL*	110		
							*LF	111					*LF	111		
				NOP	000			000			000			000		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	ASG	000		A/B	1		CL*	01		CLE	01		SEZ	SS*	SL*	
				↓	↓		CM*	10		CME	10			IN*	SZ*	
							CC*	11		CCE	11			RSS		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	IOG	000		1	H/C	HLT	000			Select Code						
				1	0	STF	001									
				1	1	CLF	001									
				1	0	SFC	010									
				1	0	SFS	011									
				A/B	1	H/C	MI*	000								
				A/B	1	H/C	LI*	101								
				A/B	1	H/C	OT*	110								
				0	1	H/C	STC	111								
				1	1	H/C	CLC	111								
				1	0	STO	001			000				001		
				1	1	CLO	001			000				001		
				1	H/C	SOC	010			000				001		
				1	H/C	SOS	011			000				001		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	EAU	000		MPY**	000	010					000			000		
				DIV**	000	100					000			000		
				DLD**	100	010					000			000		
				DST**	100	100					000			000		
				ASR	001	000				0	1					
				ASL	000	000				0	1					
				LSR	001	000				1	0					
				LSL	000	000				1	0					
				RRR	001	001				0	0					
				RRL	000	001				0	0					
Notes: * = A or B. D/I, A/B, Z/C, D/E, H/C coded: 0/1. **Second word is Memory Address.																

Basic Control System Reference Manual





CONTENTS

INTRODUCTION	v
CHAPTER 1	GENERAL DESCRIPTION 1-1
	1.1 Input/Output Subroutines 1-1
	1.2 Relocating Loader 1-2
	1.3 Prepare Control System 1-2
	1.4 Debugging System 1-3
CHAPTER 2	INPUT/OUTPUT REQUESTS 2-1
	2.1 General Calling Sequence 2-1
	2.2 Calling Sequence: Paper Tape System 2-5
	2.3 Calling Sequence: Kennedy Incremental Transport 2-9
	2.4 Calling Sequence: Magnetic Tape System 2-10
	2.5 Magnetic Tape Control Requests 2-12
	2.6 Clear Request 2-13
	2.7 Status Request 2-15
	2.8 Calling Sequence: Instruments 2-18
	2.9 Error Conditions During Execution 2-24
CHAPTER 3	RELOCATING LOADER 3-1
	3.1 External Form of Loader 3-1
	3.2 Internal Form of Loader 3-1
	3.3 Relocatable Programs 3-1
	3.4 Record Types 3-2
	3.5 Memory Allocation 3-4
	3.6 Object Program Record Processing 3-8
	3.7 Programming Considerations 3-12
	3.8 Loader Operating Procedures 3-12
CHAPTER 4	INPUT/OUTPUT DRIVERS 4-1
	4.1 General Description 4-1
	4.2 Structure 4-1
CHAPTER 5	PREPARE CONTROL SYSTEM 5-1
	5.1 Initialization Phase 5-1
	5.2 Loading of BCS Modules 5-2
	5.3 Input/Output Equipment Parameters 5-3
	5.4 Interrupt Linkage Parameters 5-6
	5.5 Processing Completion 5-8
	5.6 Operating Procedures 5-10
	5.7 Example 5-18

CHAPTER 6	DEBUGGING SYSTEM	6-1
	6.1 Operator Communication	6-1
	6.2 Control Statements	6-2
	6.3 Control Statement Error	6-5
	6.4 Halt	6-5
	6.5 Indirect Loop	6-6
	6.6 Output Formats	6-6
	6.7 Operating Procedures	6-7
	6.8 Example	6-8
APPENDIX A	HP Character Set	A-1
APPENDIX B	Equipment Table	B-1
APPENDIX C	Standard Unit Equipment Table	C-1
APPENDIX D	IOC with Output Buffering	D-1
APPENDIX E	Relocatable Tape Format	E-1
APPENDIX F	Absolute Tape Format	F-1

INTRODUCTION

The Basic Control System (BCS) provides an efficient loading and input/output control capability for relocatable programs produced by the HP Assembler, HP FORTRAN, or HP ALGOL. The system is modular in design and may be constructed to fit each user's hardware configuration.

The Basic Control System performs the following functions:

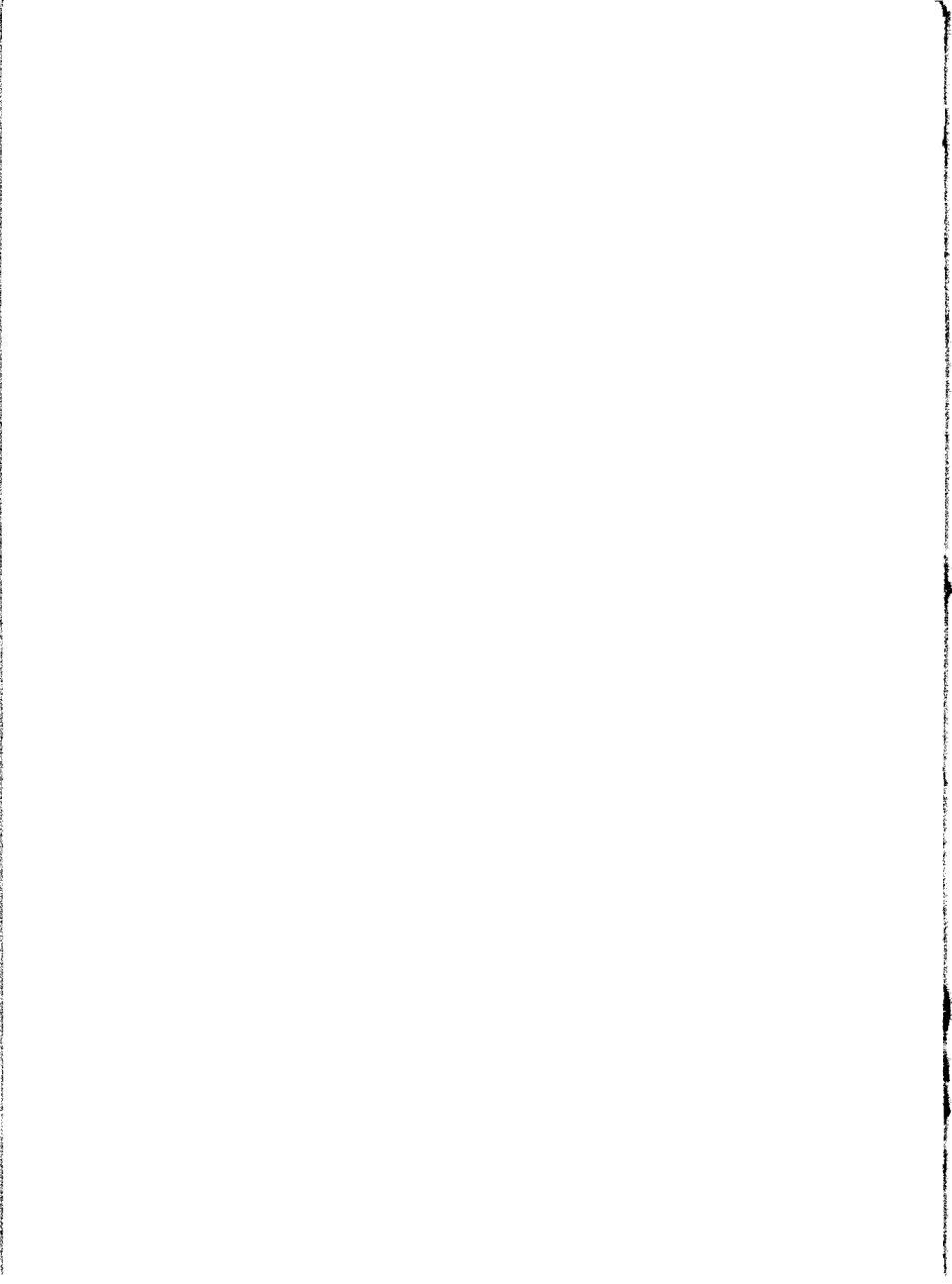
- Loads and links relocatable programs
- Creates indirect and base page addressing when necessary
- Selects and loads referenced library routines
- Processes I/O requests and services I/O interrupts

Routines associated with the Basic Control System, though not physically part of it, include

- Prepare Control System — establishes BCS configuration
- Debugging System — loaded with user program to aid in program testing.

The minimum equipment configuration required for the Basic Control System (and Prepare Control System) is as follows:

- 2116B, 2115A, or 2114A Computer with 4K memory
- 2752 Teleprinter



The Basic Control System is comprised of two distinct parts; associated with the Basic Control System are two other systems. They are as follows:

- Input/output subroutines
- Relocating Loader
- Prepare Control System
- Debugging System

The Relocating Loader loads and links relocatable object programs generated by the Assembler, FORTRAN, and ALGOL. It also links the object programs with the input/output subroutines and any library subroutines referred to in the programs. The Prepare Control System is used to adapt the Basic Control System program to a particular hardware configuration. The Debugging System is a relocatable program that BCS loads after the object program(s); with the debugging program the programmer can find errors in his program.

1.1 INPUT/OUTPUT SUBROUTINES

The input/output package consists of an Input/Output Control subroutine and driver subroutines for the peripheral devices. Input/output operations are specified as symbolic calling sequences in Assembler language. These requests are translated into object code calls to the I/O Control subroutine. The subroutine interprets the call and directs the request to the proper driver. The driver initiates the operation and returns control to the calling program. Whenever interrupt occurs, the driver temporarily resumes control to transfer the next element of data. When the operation is completed, the I/O Control subroutine makes the status of the operation available for checking by the program.

The input/output package allows device independent programming; a device is specified in terms of a unit-reference number rather than a channel number or select code. Furthermore, the user need not be concerned about how data is transmitted (by bit, by character, etc.), he need only specify the number of words or characters and the location in memory where the data is stored.

1.2 RELOCATING LOADER

The Relocating Loader loads object code programs produced by the Assembler, FORTRAN and ALGOL. The linking capability of the Loader allows the user to divide a program into several subprograms, to assemble and test each separately, and finally to execute all as one program. Object subprograms produced by the Assembler may be combined with object subprograms produced by FORTRAN and ALGOL. The subprograms are linked through symbolic entry points and external references.

The loader also provides indirect addressing whenever an operand of an instruction does not fall in the same page as that into which the instruction is being loaded. This allows a program to be designed without concern for page boundaries.

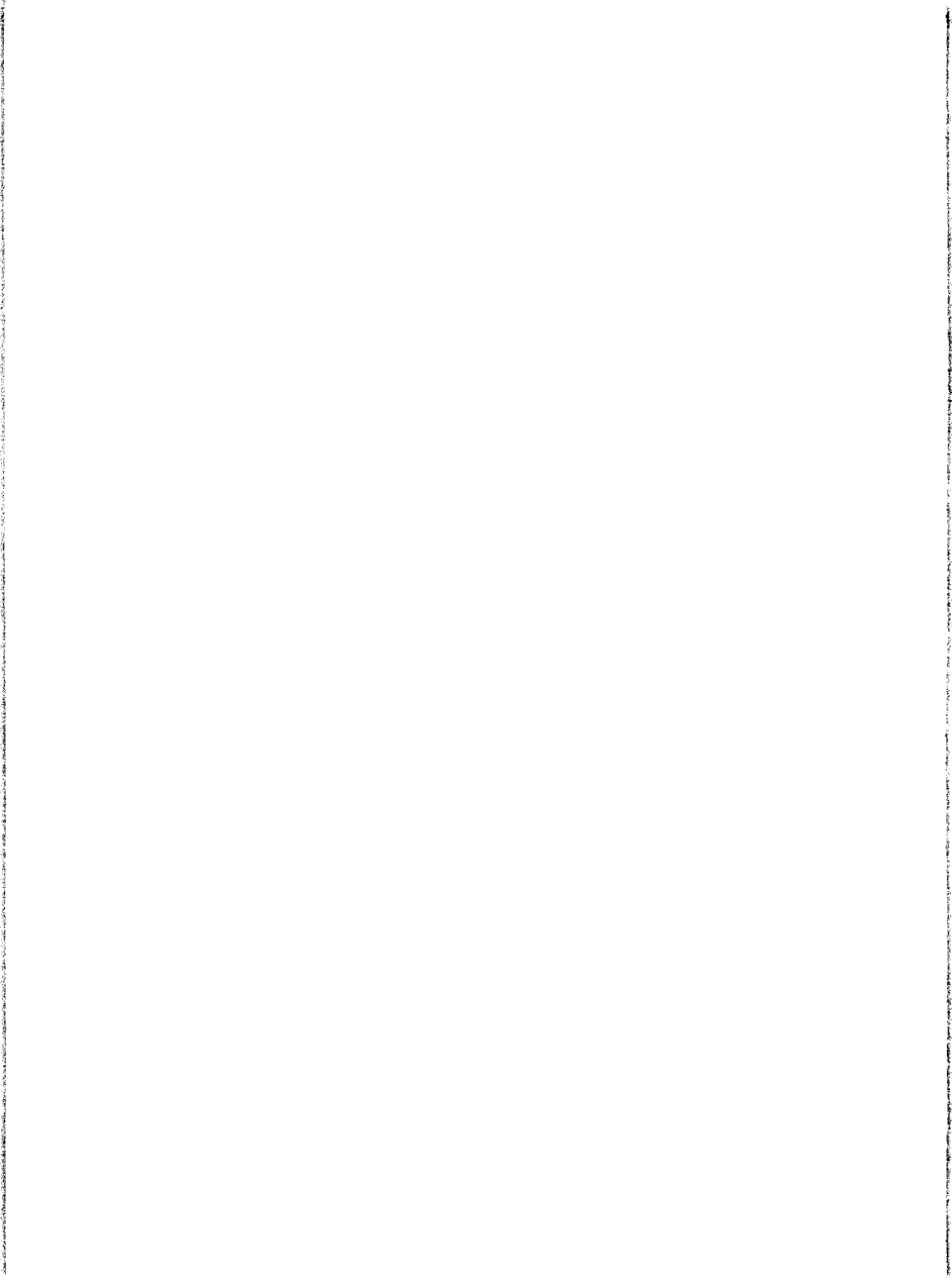
An optional feature of the Loader allows the user to obtain an absolute version of a relocatable program which may include library subroutines and segments of the Basic Control System that were referenced by the program. The process of generating the absolute program is such that instructions (not just common storage) may be allocated to the area normally occupied by the Loader. This feature may also be utilized for a program which has reached "production" status; absolute format requires less loading time because an absolute program is loaded by the Basic Binary Loader.

1.3 PREPARE CONTROL SYSTEM

Prepare Control System is a special purpose program which produces an absolute version of the Basic Control System from relocatable BCS subprograms. During the construction of the absolute BCS, the user also establishes the relationships among I/O channel numbers, drivers, interrupt entry points in the drivers, and unit-reference numbers. Prepare Control System is used when the configuration of the hardware is defined initially or whenever there is a modification or expansion to the configuration.

1.4 DEBUGGING SYSTEM

The debugging routine provides aids in program testing. Options provided by the routine will print selected areas of memory, trace portions of the program during execution, modify the contents of selected areas in memory, modify simulated computer registers, halt execution of the program at specified breakpoints, and initiate execution at any point in the program.



The Basic Control System provides the facility to request input/output operations in the form of five-word calling sequences in assembly language. The Basic Control System interprets the call, initiates the operation, and returns control to the calling program. When the data transfer is complete, the System provides status information which may be checked by the program. Interrupts which occur during or on termination of the transfer are processed entirely by the System; interrupt handling subroutines are not required in the user's program.

2.1 GENERAL CALLING SEQUENCE

The general form of the input/output request is:

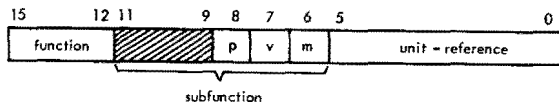
JSB	.IOC.
OCT	<function> <subfunction> <unit-reference>
{JSB} {JMP}	reject address
DEF	buffer address
{DEC} {OCT}	buffer length
	<normal return>
	.
	.
EXT	.IOC.

2.1.1 .IOC.

.IOC. is the symbolic entry point name of the input/output control subroutine within the Basic Control System. All input/output operations are requested by performing a JSB to this entry point. The input/output control subroutine returns control to the calling program at the first location following the last word of the calling sequence. Programs referring to .IOC. must declare it as an external symbol.

2.1.2 FUNCTION, SUBFUNCTION, AND UNIT-REFERENCE

The second word of the request determines the function to be performed and the unit of equipment for which the action is to be taken. In assembly language, this information may be supplied in the form of an octal constant. The bit combinations that comprise the constant are as follows:



Function

The function (bits 15-12) is the basic input/output operation; it may be either of the following:

<u>Function Name</u>	<u>Code (octal)</u>
Read	01
Write	02

Subfunction

The subfunction (bits 11-6) defines the options for certain input/output operations:

- $p = 1$ Print input: The ASCII data read from the 2752A Teleprinter is to be printed as it is received.
- $v = 1$ Variable length binary input: The value in bits 15-8 of the first word on an input paper tape indicates the length of the record (including the first word). If the value exceeds the length of the buffer, only the number of words specified as the buffer length are read. If $v = 0$, the buffer length field always determines the length of record to be transmitted. If the device does not read paper tape, the parameter is ignored.

m = 1 Mode: The data is transmitted in binary form exactly as it appears in memory or on the external device. If m = 0, the data is transmitted in ASCII or BCD format.

Unit-Reference

The value specified for the unit-reference field indicates the unit of equipment on which the operation is to be performed. The number may represent a standard unit assignment or an installation unit assignment. Standard unit numbers are as follows:

<u>Number</u>	<u>Name</u>	<u>Usual Equipment Type</u>
1	Keyboard Input	Teleprinter
2	Teleprinter Output	Teleprinter
3	Program Library	Punched Tape Reader
4	Punch Output	Tape Punch
5	Input	Punched Tape Reader
6	List Output	Teleprinter

Installation unit numbers may be in the range 78-748 with the largest value being determined by the number of units of equipment available at the installation. The particular physical unit that is referenced depends on the manner in which equipment is defined within the Basic Control System by the installation. When the Basic Control System configuration is established, an equipment table (EQT) is created. This table defines the type of equipment (Teleprinter, magnetic tape, etc.), the channel on which each unit is connected, and other related details. The ordinal of the unit's entry in this table is the value specified as a unit-reference number for an installation unit. Since numbers 1-6 are reserved as standard unit numbers, the first unit

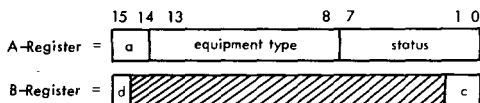
described in the table is referred to by the number 7_g; the second, 10_g; the third, 11_g; and so forth. The entries for one possible equipment table might establish the following relationships:

<u>Installation unit number</u> (ordinal)	<u>Device</u>	<u>I/O Channel</u>
7	Teleprinter	12 or 12 and 13
10	Punched Tape Reader	10
11	Tape Punch	11

The standard unit numbers are associated with physical equipment via a standard equipment table (SQT) and EQT. The SQT is a list of references to the EQT. SQT is also created by the installation when the BCS configuration is established. Each standard unit may be a separate device, or a single device may be accessed by several standard unit numbers as well as an installation unit number. (For complete details on the SQT and EQT, see Appendix B.)

2.1.3 REJECT POINT

The Basic Control System transfers control to the third word of the calling sequence if the input/output operation can not be performed. On transfer, the System provides status information which may be checked by the user's program.



The contents of the A-Register indicate the physical status of the equipment (see Status Request). The contents of the B-Register indicate the cause of the reject (bits 14-1 are zeros):

- d = 1 The device or driver subroutine is busy and therefore unavailable, or, for Kennedy 1406 Tape unit, a broken tape condition encountered.
- c = 1 A Direct Memory Access channel is not available to operate the device.
- d = c = 0 The function or subfunction selected is not legal for the device.

The content of the third word of the calling sequence is normally a JSB or a JMP to a reject address which is the start of a user subroutine designed to determine the cause of a reject and take appropriate action.

2.1.4 BUFFER STORAGE AREA

The buffer address is the location of the first word of data to be written on an output device or the first word of a block reserved for storage of data read from an input device. The length of the buffer area may be specified in terms of words or characters. If the length is given as words, the value in the buffer length field must be a positive integer; if given as characters, a negative integer.

In addition to describing the buffer area in the calling sequence, the area must also be specifically defined in the assembly language program, usually with a BSS or COM pseudo instruction.

2.2 CALLING SEQUENCE: PAPER TAPE SYSTEM

JSB	. IOC,
OCT	<function> <subfunction> <unit-reference>
{ JSB JMP }	reject address
DEF	buffer address
{ DEC OCT }	buffer length
<normal return>	
EXT	. IOC,

Allowable combinations of function and subfunction codes are as follows:

<u>Operation</u>	<u>Octal value of Bits 15-6</u>
Read ASCII record	0100
Read ASCII record and print	0104
Read binary record	0101
Read variable length binary record	0103
Write ASCII or BCD record	0200
Write binary record	0201

An illegal combination of codes is rejected.

Record Formats

ASCII Records (Paper Tape)

An ASCII record is a group of characters terminated by an end-of-record mark which consists of a carriage return, (CR), and a line feed, (LF).

For an input operation, the length of the record transmitted to the buffer is the number of characters or words designated in the request, or less if an end-of-record mark is encountered before the character or word count is exhausted. The codes for (CR) and (LF) are not transmitted to the buffer. An end-of-record mark preceding the first data character is ignored.

For an output operation, the length of the record is determined by the number of characters or words designated in the request. An end-of-record mark is supplied at the end of each output record by the input/output system.

If the last character of an output record is ←, however, the end-of-statement mark is omitted. This allows control of Teleprinter line spacing. The user may write a message (the ← is not printed) and expect the reply to be typed on the same line. The reply must be terminated with the (CR) (LF).

If a (RUB OUT) code followed by a (CR) (LF) is encountered on input from the Teleprinter or Punched Tape Reader, the current record is ignored (deleted) and the next record transmitted.†

If less than ten feed frames (all zeros) are encountered before the first data character from a paper tape input device, they are ignored. Ten feed frames are interpreted as an end-of-tape condition (see STATUS REQUEST).

Binary Records

A binary record is transmitted exactly as it appears in memory or an 8-level paper tape. The record length is specified by the number of characters or words designated in the request. The first character of a binary record must be non-zero. On input operations, less than ten feed frames preceding the first data character are ignored. Ten feed frames are interpreted as an end-of-tape condition (see STATUS REQUEST). On output, the system writes four feed frames to serve as a physical record separator.

Binary input records may be variable in length. The first word of the record contains a number in bits 15-8 specifying the length of the record (including the first word). The entire record including the word count is transmitted to the buffer. If the actual length exceeds the size of the buffer, only the number of words equivalent to the buffer length is transmitted.

† (RUB OUT) which appears on the Teleprinter keyboard is synonymous with the ASCII symbol (DEL).

2.2.2 BUFFER LENGTH

Character or word transmission may be specified for any paper tape device. The buffer length for data that may be printed on the teleprinter should be no more than 72 characters (36 words).

Examples:

Label	Column	Operation	Comment
	1	10	20
	3	11	21
	5	12	22
	7	13	23
	9	14	24
	11	15	25
	13	16	26
	15	17	27
	17	18	28
	19	19	29
	21	20	30
	23	21	31
	25	22	32
	27	23	33
	29	24	34
	31	25	35
	33	26	36
	35	27	37
	37	28	38
	39	29	39
	41	30	40
	43	31	41
	45	32	42
	47	33	43
	49	34	44
	51	35	45
	53	36	46
	55	37	47
	57	38	48
	59	39	49
	61	40	50
	63	41	51
	65	42	52
	67	43	53
	69	44	54
	71	45	55
	73	46	56
	75	47	57
	77	48	58
	79	49	59
	81	50	60
	83	51	61
	85	52	62
	87	53	63
	89	54	64
	91	55	65
	93	56	66
	95	57	67
	97	58	68
	99	59	69
	101	60	70
	103	61	71
	105	62	72
	107	63	73
	109	64	74
	111	65	75
	113	66	76
	115	67	77
	117	68	78
	119	69	79
	121	70	80
	123	71	81
	125	72	82
	127	73	83
	129	74	84
	131	75	85
	133	76	86
	135	77	87
	137	78	88
	139	79	89
	141	80	90
	143	81	91
	145	82	92
	147	83	93
	149	84	94
	151	85	95
	153	86	96
	155	87	97
	157	88	98
	159	89	99
	161	90	100
	163	91	101
	165	92	102
	167	93	103
	169	94	104
	171	95	105
	173	96	106
	175	97	107
	177	98	108
	179	99	109
	181	100	110
	183	101	111
	185	102	112
	187	103	113
	189	104	114
	191	105	115
	193	106	116
	195	107	117
	197	108	118
	199	109	119
	201	110	120
	203	111	121
	205	112	122
	207	113	123
	209	114	124
	211	115	125
	213	116	126
	215	117	127
	217	118	128
	219	119	129
	221	120	130
	223	121	131
	225	122	132
	227	123	133
	229	124	134
	231	125	135
	233	126	136
	235	127	137
	237	128	138
	239	129	139
	241	130	140
	243	131	141
	245	132	142
	247	133	143
	249	134	144
	251	135	145
	253	136	146
	255	137	147
	257	138	148
	259	139	149
	261	140	150
	263	141	151
	265	142	152
	267	143	153
	269	144	154
	271	145	155
	273	146	156
	275	147	157
	277	148	158
	279	149	159
	281	150	160
	283	151	161
	285	152	162
	287	153	163
	289	154	164
	291	155	165
	293	156	166
	295	157	167
	297	158	168
	299	159	169
	301	160	170
	303	161	171
	305	162	172
	307	163	173
	309	164	174
	311	165	175
	313	166	176
	315	167	177
	317	168	178
	319	169	179
	321	170	180
	323	171	181
	325	172	182
	327	173	183
	329	174	184
	331	175	185
	333	176	186
	335	177	187
	337	178	188
	339	179	189
	341	180	190
	343	181	191
	345	182	192
	347	183	193
	349	184	194
	351	185	195
	353	186	196
	355	187	197
	357	188	198
	359	189	199
	361	190	200
	363	191	201
	365	192	202
	367	193	203
	369	194	204
	371	195	205
	373	196	206
	375	197	207
	377	198	208
	379	199	209
	381	200	210
	383	201	211
	385	202	212
	387	203	213
	389	204	214
	391	205	215
	393	206	216
	395	207	217
	397	208	218
	399	209	219
	401	210	220
	403	211	221
	405	212	222
	407	213	223
	409	214	224
	411	215	225
	413	216	226
	415	217	227
	417	218	228
	419	219	229
	421	220	230
	423	221	231
	425	222	232
	427	223	233
	429	224	234
	431	225	235
	433	226	236
	435	227	237
	437	228	238
	439	229	239
	441	230	240
	443	231	241
	445	232	242
	447	233	243
	449	234	244
	451	235	245
	453	236	246
	455	237	247
	457	238	248
	459	239	249
	461	240	250
	463	241	251
	465	242	252
	467	243	253
	469	244	254
	471	245	255
	473	246	256
	475	247	257
	477	248	258
	479	249	259
	481	250	260
	483	251	261
	485	252	262
	487	253	263
	489	254	264
	491	255	265
	493	256	266
	495	257	267
	497	258	268
	499	259	269
	501	260	270
	503	261	271
	505	262	272
	507	263	273
	509	264	274
	511	265	275
	513	266	276
	515	267	277
	517	268	278
	519	269	279
	521	270	280
	523	271	281
	525	272	282
	527	273	283
	529	274	284
	531	275	285
	533	276	286
	535	277	287
	537	278	288
	539	279	289
	541	280	290
	543	281	291
	545	282	292
	547	283	293
	549	284	294
	551	285	295
	553	286	296
	555	287	297
	557	288	298
	559	289	299
	561	290	300
	563	291	301
	565	292	302
	567	293	303
	569	294	304
	571	295	305
	573	296	306
	575	297	307
	577	298	308
	579	299	309
	581	300	310
	583	301	311
	585	302	312
	587	303	313
	589	304	314
	591	305	315
	593	306	316
	595	307	317
	597	308	318
	599	309	319
	601	310	320
	603	311	321
	605	312	322
	607	313	323
	609	314	324
	611	315	325
	613	316	326
	615	317	327
	617	318	328
	619	319	329
	621	320	330
	623	321	331
	625	322	332
	627	323	333
	629	324	334
	631	325	335
	633	326	336
	635	327	337
	637	328	338
	639	329	339
	641	330	340
	643	331	341
	645	332	342
	647	333	343
	649	334	344
	651	335	345
	653	336	346
	655	337	347
	657	338	348
	659	339	349
	661	340	350
	663	341	351
	665	342	352
	667	343	353
	669	344	354
	671	345	355
	673	346	356
	675	347	357
	677	348	358
	679	349	359
	681	350	360
	683	351	361
	685	352	362
	687	353	363
	689	354	364
	691	355	365
	693	356	366
	695	357	367
	697	358	368
	699	359	369
	701	360	370
	703	361	371
	705	362	372
	707	363	373
	709	364	374
	711	365	375
	713	366	376
	715	367	377
	717	368	378
	719	369	379
	721	370	380
	723	371	381
	725	372	382
	727	373	383
	729		

2.3 CALLING SEQUENCE:

KENNEDY INCREMENTAL TRANSPORT

JSB	. IOC.
{ OCT }	<function> <subfunction> <unit-reference>
{ JSB JMP }	reject address
DEF	buffer address
{ DEC OCT }	buffer length
EXT	. IOC.

Function and Subfunction

Allowable function codes for the 1406 Kennedy Incremental Tape Transport are as follows:

WRITE (ASCII Mode only)	0200
WRITE End-of-file	0301
CLEAR	0000

Record Formats

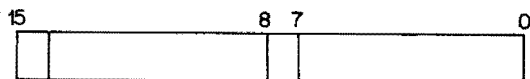
Binary Coded Decimal Records (Magnetic Tape)

A BCD record is a group of BCD characters terminated (on magnetic tape) by a record gap. A request to write a BCD record results in the translation of each 7-level ASCII character in the buffer area into a 6-level BCD character on magnetic tape. The translation process does not alter the original contents of the buffer.

The length of the record is determined by the number of characters or words designated in the request. A record gap is supplied at the end of each record by the input/output system.

If the last character in the buffer area is -, however, the record gap is omitted. The - is not written on tape.

A WRITE request specifying a buffer length of zero causes a record gap only to be written.



2nd ASCII character

1st ASCII character

TRANSLATION: from 7-level ASCII to 6-level BCD

1st Tape character

2nd Tape character

X	X	X	X	X	X
X	X	X	X	X	X

2.4 CALLING SEQUENCE: MAGNETIC TAPE SYSTEM

JSB . IOC.

OCT <function> <subfunction> <unit-reference>

{ JSB
JMP } reject address

DEF buffer address

{ DEC
OCT } buffer length

⋮

EXT . IOC.

2.4.1 2020A Magnetic Tape Unit

All allowable combinations of function and subfunction codes are as follows:

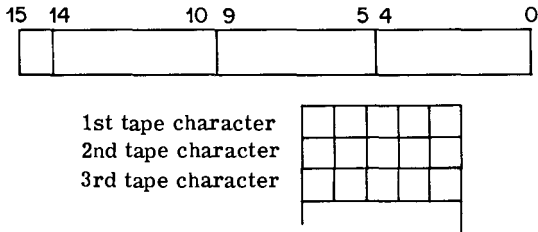
Operation	Octal value of Bit 15-6
Read BCD record and convert to ASCII	0100
Read binary record	0101
Write BCD record after converting from ASCII	0200
Write binary record	0201

RECORD FORMATS

BINARY RECORDS (MAGNETIC TAPE)

A binary record on magnetic tape is a group of 6-level tape "characters" recorded in odd parity and terminated by a record gap.† The record length is determined by the number of characters or words in the buffer as designated in the request.

Each computer word is translated into three tape "characters" (and vice versa) as follows:



For output operations, the minimum buffer length is 3 computer words.

BINARY CODED DECIMAL RECORDS

A BCD record on magnetic tape is a group of BCD characters recorded in even parity and terminated by a record gap. (See Appendix A for BCD character set.) A request to write a BCD record results in the translation of each 7-level ASCII character in the buffer area into a 6-level BCD character on magnetic tape. A request to read a BCD record results in the translation of each BCD character into an ASCII character after the block has been read.

The length of the record may not be more than 120 characters. A record gap is supplied at the end of each record.

† Odd parity: a seventh bit is recorded on tape if the total of the bits in the six levels is an even number.

Even parity: a seventh bit is recorded on tape if the total of the bits in the six levels is an odd number.

A WRITE request for the HP 2020A Magnetic Tape Unit must have a minimum buffer length of seven ASCII characters (four words). If less than seven characters are specified, spaces will be added to fill the seven characters.

2.4.2 3030 MAGNETIC TAPE UNIT

FUNCTION AND SUBFUNCTION

All allowable combinations of function and subfunction codes are as follows:

READ	0101	Binary only
WRITE	0201	Binary only
WRITE FILE MARK	0301	
SPACE	0302	Backspace one record
	0303	Forward space one record
REWIND	0304	Rewind to load point - Ready
	0305	Rewind to load point - Standby

Buffer Length

Character transmission is not applicable since the transmission is via a DMA channel. The minimum data block is twelve tape characters. Output blocks with a block length less than twelve characters are padded with zeroes.

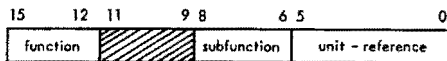
2.5 MAGNETIC TAPE CONTROL REQUESTS

A request directed to .IOC. may also control the positioning of a reel on a magnetic tape device. The calling sequence is similar to the input/output request, but consists of only three words:

```

EXT      .IOC.
:
JSB      .IOC.
OCT      <function> <subfunction> <unit-reference>
{ JSB    }
{ JMP    } reject address
<normal return>
```

The second word of the request has the following composition:



The function defines the calling sequence as a tape positioning request:

<u>Function Name</u>	<u>Code (octal)</u>
Position Tape	03

The subfunction defines the type of positioning:

<u>Subfunction (octal)</u>	<u>Operation</u>
0	Dynamic tape status
1	Write End-of-File
2	Backspace one record
3	Forward space one record
4	Rewind
5	Rewind and Standby

As soon as tape movement operations (Rewind, and Standby) are initiated, the device is considered to be available; the "a" field of a status reply (0400 code) is set to 00. The input/output driver is thus free to process requests for other devices. To obtain the actual status of the device when one of these commands has been issued, the Dynamic tape status request is used. If the tape movement operation is still in progress the "a" field is set to 10.

2.6 CLEAR REQUEST

The clear request terminates a previously issued input or output operation before all data is transmitted. It has the following form:

```

EXT      .IOC.
:
:
JSB      .IOC.
OCT      <function> <unit-reference>
<normal return>

```

The second word consists of the following:



The function has the following value:

<u>Function Name</u>	<u>Code (octal)</u>
Clear	00

The only other parameter required is the unit-reference number. If the unit-reference number is specified as 00 (i.e., the second word of the calling sequence is OCT 0), all previous input and output operations are terminated. This request, the system clear request, makes all devices available for the initiation of a new operation. On return from a system clear request, the contents of the A- and B-Registers are meaningless.

Example:

Label	Operation	Comments	Comments
READM	JSB .IOC.	READ AND PRINT A MESSAGE OF ONE	
	OCT 10401	LINE FROM THE TELEPRINTER. WHEN	
	JMP REJ	CONTROL RETURNS AFTER INITIATING	
	DEF MSG	THE REQUEST, THE JSB MIGHT	
	DEC 36	TRANSFER TO A SUBROUTINE WHICH	
	JSB TIMER	COULD CHECK THE TIME ALLOWED	
		FOR A MESSAGE TO BE COMPLETED.	
CLRDR	JSB .IOC.	IF THE MESSAGE IS NOT FURNISHED	
	OCT 1	WITHIN A SPECIFIC TIME LIMIT, THE	
		REQUEST IS CLEARED BY THE SECOND	
		REQUEST TO .IOC.	

2.7 STATUS REQUEST

A request may be directed to .IOC. to determine the status of a previous input/output request or to determine the physical status of one or all units of equipment. The request has the following form:

```
JSB      .IOC.
OCT      <function> <unit-reference>
<normal return>
```

The second word of the request has the following form:

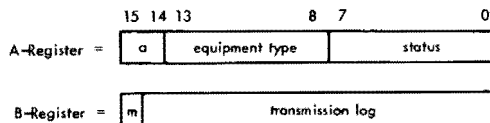


The function has the following value:

<u>Function Name</u>	<u>Code (octal)</u>
Status	04

The calling sequence requires no other parameters. A reject location is not necessary since the status information is always available. If the unit-reference number is specified as 00 (i.e., the second word on the calling sequence is OCT 40000), the request is interpreted as a system request.

If information is requested for a single unit, the Basic Control System returns to the location immediately following the request with the status information in the A and B registers:



- a Availability of device:
- 0 The device is available; the previous operation is complete.
- 1 The device is available; the previous operation is complete but a transmission error has been detected.
- 2 The device is not available for another request; the operation is in progress.

equipment type

This field contains a 6-bit code that identifies the device referenced:

- 00-07 - Paper Tape devices
 - 00 2752A Teleprinter
 - 01 2737A Punched Tape Reader
 - 02 2753A Tape Punch
- 10-17 - Unit Record devices
- 20-37 - Magnetic Tape and Mass Storage devices
 - 20 Kennedy 1406 Incremental Tape Transport
 - 21 HP 2020A Magnetic Tape Unit
 - 22 HP 3030A Magnetic Tape Unit
- 40-77 - Instrumentation devices
 - 40 Data Source Interface
 - 41 DVM Programmer
 - 42 Scanner Programmer
 - 43 Time Base Generator

status

The status field indicates the actual status of the device when the data transmission is complete. The contents depend on the type of device referenced:

Teleprinter reader or Punched Tape Reader:

Bits 7-0

Condition

xx1xxxxx End-of-Tape (10 Feed Frames)

Tape Punch:

Bits 7-0

Condition

xx1xxxxx Tape supply low

Kennedy 1406 Incremental Tape Transport:

Bits 7-0

Condition

xx1xxxxx End-of-Tape mark sensed

xxxx1xxx Broken tape; no tape on write head

xxxxxxx1 Device busy

HP 2020A and 3030A Magnetic Tape Units:

Bits 7-0

Condition

1xxxxxxx end-of-file record (17₈)

encountered while reading, forward spacing, or backward spacing.

x1xxxxxx start-of-tape marker sensed

xx1xxxxx end-of-tape marker sensed

xxx1xxx timing error on read/write

xxxx1xxx I/O request rejected:

- a. tape motion required but controller busy
- b. backward tape motion required but tape at load point
- c. write request given but reel does not have write enable ring.

xxxxx1xx Reel does not have write enable ring or tape unit is rewinding.

xxxxxxx1x Parity error on read/write

xxxxxxx1 Tape in motion

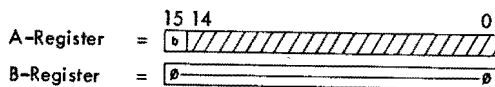
m This bit defines the mode of the data transmission:

0 ASCII or BCD

1 Binary

transmission log This field is a log of the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in calling sequence. The value is stored in this field only when the request is completed, therefore, when all data is transmitted or when a transmission error is detected.

If a system status request is made, the information in the A and B registers is as follows:



b System Status
 0 No device is busy
 1 At least one device is busy

2.8 CALLING SEQUENCE: INSTRUMENTS

2.8.1 DATA SOURCE INTERFACE

A binary output operation causes the removal of "hold-off." The calling influence is as below:

JSB . IOC.
 OCT <function> <subfunction> <unit-reference>
 { JSB }
 { JMP } reject address
 OCT 0 dummy buffer
 OCT 0 buffer length

Example:

[illegible]

A binary input operation must have a 2-word buffer. Thirty-two bits (8 BCD characters) are read directly into the 2-word buffer.

JSB	, IOC,
OCT	<function> <subfunction> <unit-reference>
{ JSB JMB }	reject address
DEF	buffer address
DEC	-32
⋮	
buffer address BSS	2

Example:

[illegible]

An ASCII input operation must have an 8 word buffer. 8 BCD characters are converted into 16 ASCII characters in the following format:

$$\text{rf. } d_5 d_4 d_3 d_2 d_1 d_0 E - ss \wedge gg$$

r range - a negative power of 10

f function

d_5-d_0 six digit data value

E-ss range expressed as an exponent of two digits

^^ two blanks

gg function expressed as a two-digit number

JSB , IOC,

OCT <function> <subfunction> <unit-reference>

JMP	reject address
ICD	

DEFINITION 1. Let \mathcal{A} be a \mathcal{C}^* -algebra and let \mathcal{K} be a \mathcal{C}^* -algebra. A \mathcal{C}^* -algebra \mathcal{B} is called a \mathcal{C}^* -algebra extension of \mathcal{A} by \mathcal{K} if there is a \mathcal{C}^* -algebra isomorphism $\phi: \mathcal{A} \rightarrow \mathcal{B}$ such that $\phi(\mathcal{A})$ is a closed ideal of \mathcal{B} and $\mathcal{B}/\phi(\mathcal{A}) \cong \mathcal{K}$.

152 12

•

•

BSS 8

buffer address	BSS	8
----------------	-----	---

Example:

Label	Comments	Operation	Comments	Comments
		JSB .IOC.		
		OCT 10015	READ ON CHANNEL 15	
		JMP REJAD		
		DEF BUFF		
		DEC -16		
		.		
		.		
		.		
BUFF	BSS	8		

2.8.2 DIGITAL VOLTMETER PROGRAMMER

A write request for the Digital Voltmeter Programmer requires that a one-word buffer be specified. This word contains the voltmeter program: sample period (bit 7-6), function (bits 5-3), and range (bits 2-0). If bit 15 contains a 1, encode command is sent to the Voltmeter (bit 15 always be 0 if the configuration includes a Scanner).

```

JSB      .IOC.
OCT      <function> <subfunction> <unit-reference>
JSB
JMP      reject address
DEF      buffer address
OCT      1
      :
      :
buffer address OCT      voltmeter program
  
```


Example:

[illegible]

2.8.4 INSTRUMENT CLEAR REQUEST:

A clear request on one of the instrument drivers follows the standard form:

JSB IOC.

OCT <function> <unit-reference>

<normal return>

where the function code = 00.

The request will result in the following conditions:

Data Source Interface – A clear request causes no action. It is included for compatibility only.

Digital Voltmeter Programmer - A clear request to this driver will remove the present program from the DVM but the program will not be destroyed.

Crossbar Scanner - A clear request will inhibit the STEP or RESET command on the Scanner programmer driver.

INSTRUMENT STATUS REQUEST:

No status information is available from the instrument drivers.

2.9 ERROR CONDITIONS DURING EXECUTION

Illegal conditions encountered during .IOC. request processing are termed irrecoverable and cause a halt.† Diagnostic information is displayed in the A- and B-Registers at the time of the halt.

The B-Register contains the absolute location of the JSB instruction of the request call containing the illegal condition.

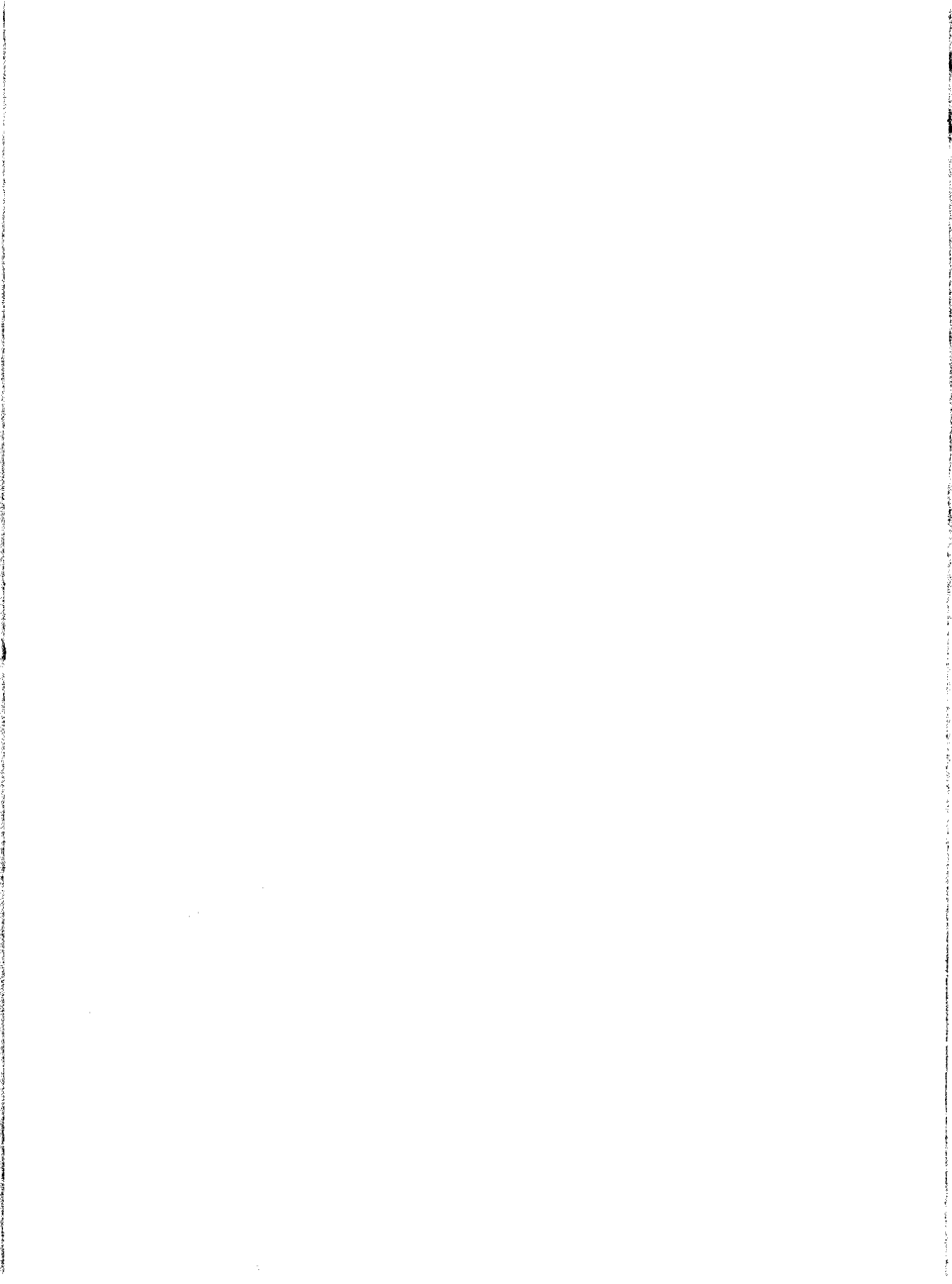
The A-Register contains a code defining the illegal condition:

<u>A-Register</u>	<u>Explanation</u>
000000	Illegal request code.
000001	Illegal unit-reference number in request.
000002	The Standard unit requested is not defined as a particular device in the Equipment Table.

†The halt is at the absolute location assigned to the symbol IOERR during Prepare Control System processing.

Examples:

Line	Operation	Comment
1	EXT .IOC.	DECLARE .IOC. AS EXTERNAL
2	INARA BSS 10	SYMBOL. RESERVE STORAGE
3	.	AREA.
4	.	
5	READM JSB .IOC.	READ AN ASCII RECORD FROM
6	OCT 10015	UNIT-REFERENCE NUMBER 15 AND
7	JMP RJCT	STORE AT LOCATION INARA.
8	DEF INARA	
9	DEC 10	
10	.	
11	.	
12	STATM JSB .IOC.	CHECK STATUS OF READ REQUEST.
13	OCT 40015	IF INITIAL BIT 15 SET, UNIT 15
14	SSA	IS BUSY; LOOP ON STATUS REQUEST
15	JMP STATM	UNTIL OPERATION IS COMPLETE.
16	RAL	CHECK INITIAL BIT 14. IF SET,
17	SSA	TRANSFER TO END-OF-TAPE CHECK.
18	JMP EOT	IF INITIAL BIT 5 SET, PERFORM
19	JMP PROCS	ENDING PROCESS AT ENDPR.
20	EOT ALF, ALF	IF NOT SET, TRANSFER TO TERMINA-
21	RAL	TION PROCEDURE AT ABORT. IF
22	SSA	REQUEST COMPLETED, CONTINUE
23	JMP ENDPR	PROCESSING AT PROCS.
24	JMP ABORT	
25	.	
26	.	
27	RJCT SSB	DETERMINE CAUSE OF REJECT
28	JMP READM	CONDITION. IF THE DEVICE OR
29	JMP ABORT	DRIVER IS BUSY, LOOP ON REQUEST
30		UNTIL AVAILABLE. IF REJECTED FOR
31		ANY OTHER REASON, TERMINATE THE
32		PROGRAM AT ABORT.



The Loader is the module of the Basic Control System that provides the capability of loading, linking, and initiating the execution of relocatable object programs produced by the Assembler, FORTRAN, and ALGOL. It is available in 4K and 8K versions. ALGOL programs and the Program Library stored on magnetic tape require the 8K loader.

3.1 EXTERNAL FORM OF LOADER

The Loader is stored in an absolute record format on an external medium with the Input/Output Control subroutine, .IOC., and the equipment driver subroutines. It is loaded by the Basic Binary Loader. The external medium is determined by the type of device assigned as the Standard Input unit. For the Punched Tape Reader or the Teleprinter, the medium is 8-level paper tape.

3.2 INTERNAL FORM OF LOADER

The Loader is located in high-numbered memory along with the Input/Output Control subroutine and the equipment driver subroutines. The Loader uses .IOC. for input/output operations; it refers to the Standard input and output units. The binary object program is read from the Standard Input unit; comments to the user (e.g., Loader diagnostics) are written on the Teleprinter Output unit; and library routines referenced by the object program are assumed to be on the Program Library unit.

3.3 RELOCATABLE PROGRAMS

The process of assembling or compiling a set of symbolic source program statements may be specified to result in the generation of a relocatable object program. A relocatable program assumes a starting location of 00000. Location 00000 is termed the relative, or relocatable origin. The absolute origin (also termed the relocation base) of a relocatable program is

determined by the Loader. The value of the absolute origin is added to the zero-relative value of each operand address to obtain the absolute operand address. The absolute origin, and thus the values of every operand address, may vary each time the program is loaded.

A relocatable program may be made up of several independently assembled or compiled subprograms. Each of the subprograms would have a relative origin of 00000. Each subprogram is then assigned a unique absolute origin upon being loaded. Subprograms executed as a single program may be loaded in any order. The absolute origins will differ whenever the order of loading differs.

The operand values produced by the Assembler, FORTRAN, or ALGOL may be program relocatable, base page relocatable, or common relocatable. Each of these segments of the program has a separate relocation base or origin. Operands that are references to locations in the main portion of the program are incremented by the program relocation base; those referring to the base page, by the base page relocation base; and those referring to common storage, by the common relocation base.

If the Loader encounters an operand that is a reference to a location in a page other than the "current" page, a link is established through the base page. A word in the base page is allocated to contain the full 15-bit address of the referenced location. The address of the word in the base page is then substituted as an indirect address in the instruction in the "current" page. If other similar references are made to the same location, they are linked through the same word in the base page.

3.4 RECORD TYPES

The Loader processes three to five record types for a program. These record types are produced by the Assembler, FORTRAN, or ALGOL in the following sequence:

NAM	Name record
ENT	Entry point record
EXT	External name record
DBL	Data block record
END	End record

The NAM, DBL, and END records exist for every object program; ENT and EXT appear only if the corresponding pseudo instructions are used in the source program.

NAM

The NAM record contains the name of the program and the length of the main, base page, and common segments. The NAM record signifies the beginning of the object program.

ENT

The ENT record defines the names of 1 to 14 entry points within this program. Each of the four-word entries in the record contains the name, the relocatable address of the name; and an indicator which specifies whether the address is program or base page relocatable.

EXT

The EXT record contains from 1 to 19 three-word entries which specify the external references defined in the program. The three words allow a maximum of five ASCII characters for the symbol and a number used by the Loader to identify the symbol.

DBL

A DBL record contains 1 to 45 words of the object program. It indicates the relative starting address for the string of words and whether this portion of the object code is part of the main program or base page segment. For each of the words there is also a relocation indicator which defines the relocation base to be applied to each operand value. Possible relocation factors are:

Absolute	Operand is an absolute expression or constant. There is no relocation base.
15-bit Program Relocatable	Operand is a 15-bit value to which is added the program relocation base.
15-bit Base Page Relocatable	Operand is a 15-bit value to which is added the base page relocation base.

15-bit Common Relocatable	Operand is a 15-bit value to which is added the common relocation base.
External Symbol Reference	Operand is a reference to an external symbol. Value is supplied when the Loader determines the absolute location of the linkage word in the Base Page which contains the 15-bit address of the related entry point.
Memory Reference Instruction	A memory reference instruction in the form of a two-word group which consists of the instruction code, a full 15-bit operand address, and a relocation indicator for the operand address. The relocation indicator can define the operand address to be program, base page, or common relocatable.

END

The END record terminates the block of records in an object program. The END record may contain a 15-bit address which is the location to which control is transferred by the Loader to begin program execution.

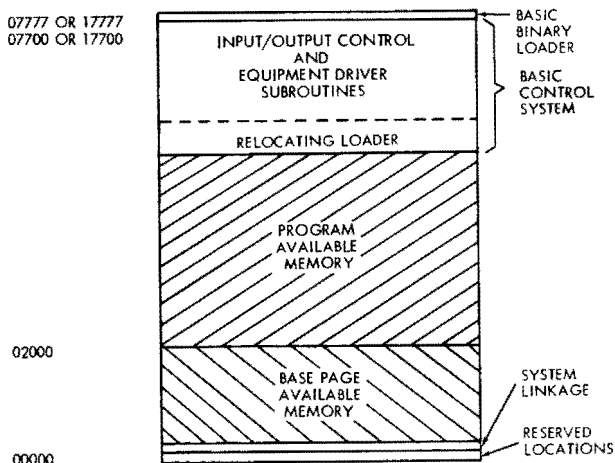
3.5 MEMORY ALLOCATION

The Loader loads the object program into available memory. Available memory is defined as that area of memory not allocated for hardware and system usage. Available memory is divided into two segments:

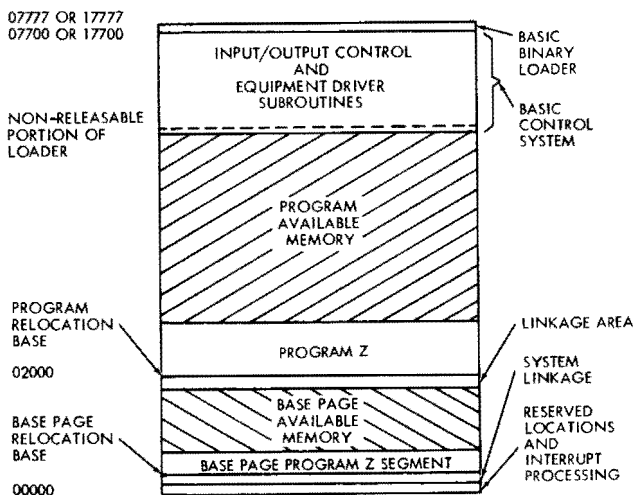
Available Memory in Base Page – used for the operand linkage area, program blocks originated into the Base Page by the Assembler pseudo instruction ORB, and for program blocks assigned to the Base Page by the Loader when the amount of program available memory is insufficient.

Program Available Memory – used for the main body of the program and may be used by the common block should the area used by the Loader be insufficient.

Prior to loading the object program, memory is allocated as follows:



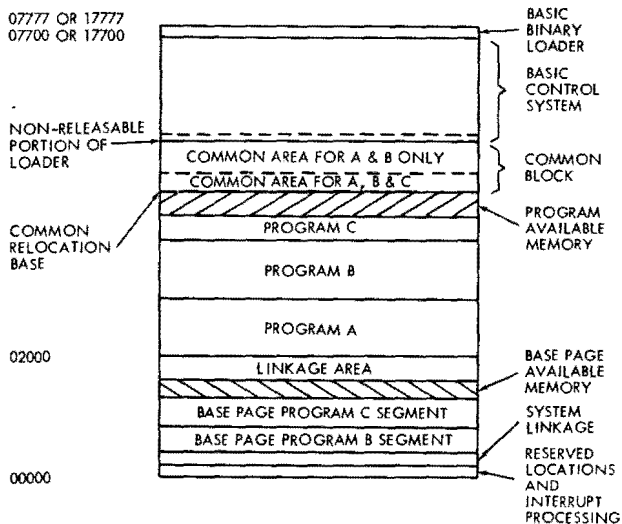
Assuming Program Z is to be loaded and executed – after loading, the memory might be allocated as follows:



Options selected during PCS processing can define the equipment driver subroutines and other system routines as relocatable programs. If selected, these routines would be allocated to the available memory areas, and the length of the absolute segment of BCS reduced accordingly.

If several programs are to be loaded and executed together, the following might occur:

Assume three programs, A, B, and C, comprise a running program. Programs A and B share a common block, a portion of which is also shared by C. Programs B and C contain segments which are designated to be allocated to the Base Page. Allocation is as follows:



Common Block Allocation

The first common length declaration (i.e., the first program containing a common segment) processed by the Loader establishes the total common storage allocation in high memory overlaying the major portion of the area occupied by the Loader. Subsequent programs must contain common length declarations

which are less than or equal to the length of the first declaration.

To allocate the common area, the Loader subtracts the total length of the block from the address of the last releasable word in the Loader. The resulting memory address +1 is the origin of the common block. This value is used throughout the entire loading process as the common relocation base.

Program Storage

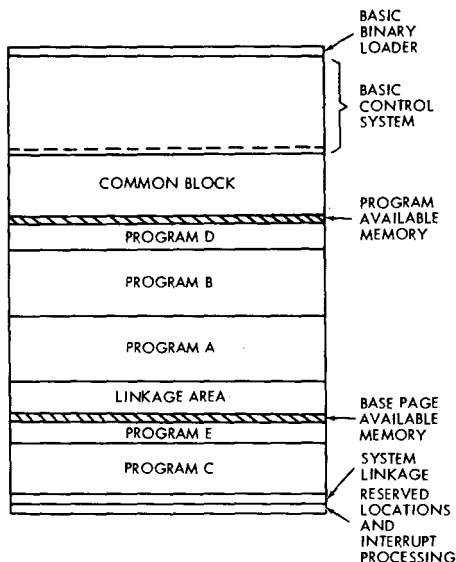
The program length is compared with the amount of available memory. If sufficient space is available, the program is loaded and the upper and lower bounds recorded. If the program has a Base Page segment, or if the program consists entirely of coding to be stored in the Base Page, the length of the segment is compared to the amount of available base page memory. If there is enough space in this area, the segment is loaded and the bounds recorded. The initial main program segment is usually originated at absolute location 02000 (page 1, Module 0). The initial Base Page segment is usually originated immediately following the area set aside for Reserved Locations, Interrupt Processing, and System Linkage. Subsequent main program and Base Page segments are loaded into the next available higher numbered areas contiguous to the previously loaded segments.

Providing the Memory Allocation List option is selected (see Operating Procedures) the name of each program, its upper and lower bounds, and its Base Page upper and lower bounds are printed after the program is loaded. The format is as follows:

```
<program name>
11111 uuuuu (main program bounds)
11111 uuuuu (Base Page bounds)
```

If the Loader finds that the main program segment about to be loaded can not fit in the memory area available for the main segment, it compares the segment's length to the length of available memory in the Base Page. If there is sufficient space, the main segment will be loaded in the Base Page. The next segment will be loaded in the main program area if it will fit, or in the Base Page if not (providing there is sufficient space in the Base Page).

For example, assume that several programs to be loaded in sequence A, B, C, D, E, and have sizes such that they can not all fit in the main program available memory.



3.6 OBJECT PROGRAM RECORD PROCESSING

ENT/EXT Record Processing

The Loader constructs and maintains a Loader Symbol Table which contains entry points and external symbols which are declared in the programs and entry point names of any BCS system subroutines that have been defined as relocatable. As each entry point is encountered its relocated (absolute) address is recorded in the table. As each external reference is processed, a link word is established in the Base Page. The gen-

eral processing of the entries in an ENT and EXT record involves searching the Table to locate a match between the symbols. When a match is found, the absolute entry point address is stored in the Base Page link word.

The Loader assumes that there is a user program, BCS system routine, or Program Library routine entry point for every external reference. If none exists, the external reference is undefined and considered to be in error. A list of undefined external symbols is printed at the end of the loading operation. If duplicate entry points are detected, a diagnostic is issued. A user entry point duplicated by a Program Library routine results in the library routine being ignored.

Each entry in the Loader Symbol Table occupies five words. The Table is positioned before the beginning of the Loader and extends backwards toward low-numbered memory. If sufficient space is not available in the main program portion of memory to store a five-word entry, a diagnostic message is issued and the loading operation is terminated.

DBL Record Processing

A load address for the data or instruction words in a DBL record is relocated by adding either the program relocation base or the base page relocation base. The resulting value is the absolute address for storing the first word. The second word is stored at address +1, the third at address +2 and so forth. A relocation base is added to each operand address as specified by the relocation indicator.

The processing for an external reference word involves a search of the Loader Symbol Table for the related entry. When found, the address of the link location in the Base Page is extracted and stored as an indirect address in the instruction.

When a memory reference instruction is processed, the Loader first applies the proper relocation base, (program, base page, or common) to the 15-bit operand address. If the resulting absolute operand address references the Base Page, the address (bits 09-00) is set into the operand field and the instruction is stored in memory at the current load address. When the absolute operand address and the current load address are in the same page, the operand address is truncated to bits 09-00 and set as the instruction operand address. If

the operand address is in a page other than the current load address page, the operand address is stored in the Linkage area of the Base Page and a reference to this location set as an indirect address in the operand field of the instruction.

A memory overflow condition can occur when insufficient space is available in the base page to allocate a linkage word. A diagnostic message is issued and the loading operation is terminated.

END Record Processing

When an END record is encountered, the Loader determines if it contains a transfer of control address. If it does, the address is saved.

If loading is from the Program Library and there exist no undefined external references, the End-Of-Loading operation is performed.

If loading is from the Standard Input unit or Program Library unit and if undefined external references exist, the Loader requests the next record. If the next record is a NAM record, processing of the next program begins. If the result of the request is an End-of-Information indication, an End Condition exists.

Program Library Loading

Loading from the Program Library differs from loading of user programs in that only those programs in the Library which contain entry points matching undefined external symbols in the Loader Symbol Table will be loaded. After each program is loaded from the Library, the Loader Symbol Table is checked for undefined symbols. If none exist, the loading operation is complete and the program is ready to be executed.

End Condition

When the Loader requests input and no data is available on the input device an End Condition exists. The Loader acknowledges this condition by writing the message "LOAD" on the Teleprinter Output device. The user responds to this message by

setting switches 2-0 of the Switch Register (see Operating Procedures). Four replies are available:

- a. Load next program from Standard Input unit. Relocatable BCS system subroutines are considered to be part of the program and must be loaded from the Standard Input unit (unless they are made part of Program Library tape).
- b. All programs are loaded; proceed to the end-of-loading operation.
- c. Terminate loading operation. This forces program execution even though there may be undefined external references.
- d. Load from Program Library unit; all user programs are loaded.

End-of-Loading Operation

The end of loading is signaled by the second or fourth response to an End Condition. The Loader then searches the Loader Symbol Table for any undefined external references. Any such undefined external symbols are written on the Teleprinter Output unit and the "LOAD" message is repeated.

When the loading operation is completed, or when the user has requested termination of the Loading process, the Loader produces a memory allocation list. (This list may be omitted; see Operating Procedures.) The format of the list is as follows:

<symbol 1> aaaaa

<symbol 2> aaaaa

.

.

<symbol n> aaaaa

The symbols are the entry points in the user's program, the Basic Control System, or the Program library and the a's are their absolute addresses.

If a common block was allocated, the lower and upper bounds of the block are listed as follows:

*COM lllll uuuuu

The bounds of the Linkage Area are listed as follows:

*LINKS lllll uuuuu

The l's are the absolute lower bounds and the u's are the absolute upper bounds.

3.7 PROGRAMMING CONSIDERATIONS

When a program has been completely loaded, its execution is initiated by performing a Jump Subroutine to the transfer address (from the last END record containing an address). The initial contents of the transfer address should be a NOP, OCT 0, etc., not the first executable instruction of the program.

3.8 LOADER OPERATING PROCEDURES

The exact operating procedures for the loader depend on the available hardware configuration and the construction of the Basic Control System through use of the Prepare Control System routine. The user should know the assignment of input/output equipment and memory size before using the Loader.†

†As established when configuring BCS.

Loading Options

The Basic Control System Loader is designed to load one or more tapes containing relocatable programs. The message "LOAD" is typed when an end-of-tape condition is encountered. The user then loads the next tape; indicates loading from the program library, specifies that loading is complete, etc. When all programs are loaded and no undefined external references remain, the Loader types the message "LST" allowing the user to bypass part of the Memory Allocation List. Following the response, the Loader types the message "RUN". The user then initiates program execution.

Memory Allocation List

A memory Allocation List may be obtained for the programs being loaded. The list may include the name, main program bounds, and Base Page bounds for each of the programs. This portion of the List may be followed (at the completion of the loading operation) by a list of all entry points and their absolute addresses, the bounds of the common block, and the bounds of the linkage area. The setting of Switch 15 determines the contents of the List.

To obtain the bounds for each program on a tape, Switch 15 must be set to 0 before the tape is loaded (in response to the "LOAD" message). To bypass the program bounds listing, set Switch 15 to 1 before loading the tape. The switch setting may be altered whenever the "LOAD" message is typed.

To obtain the entry point list, the common bounds, and the linkage area bounds, set Switch 15 to 0 in response to the message "LIST", which is printed after all programs are loaded. To bypass this portion of the list, set Switch 15 to 1.

Absolute Binary Output

The user may specify that an absolute binary tape be punched. This option may be selected when it is necessary to utilize the area occupied by the Loader or when an absolute version is desired for "production stage" programs. The process involves a simulated loading operation, however, the absolute program is punched on tape rather than being loaded.

The absolute records produced consist of the relocated programs (including all programs loaded from the Program Library), the Linkage area, all referenced segments of the Basic Control System. These might include:

Input/Output control subroutine (.IOC.)
All input/output equipment drivers
Other BCS system subroutines
Memory Table (.MEM.)
System Linkage Area
Interrupt Processing area
Absolute location 2 and 3

In addition, the Loader Symbol Table, the common and linkage area bounds are punched in ASCII format on the end of the binary tape. Ten inches of feed frames separate the binary instructions and the ASCII data. This feature provides a record of the memory allocation for the absolute program.

At the completion of the "loading" process the Loader types the message "END".

To execute the program, it must be loaded using the Basic Binary Loader. To initiate execution, set 000002 into the P-Register and press RUN. The Loader has stored the transfer address of the program in locations 2 and 3 as follows:

2 contains JMP 3,I
3 contains < transfer address >

Separation of List and Binary Output

If the absolute binary output option is selected and the Teleprinter is used as both a list and punch device, the Loader halts before and after each line is printed to avoid punching the line and altering the binary output.

The halts and related procedures are as follows:

<u>T-Register Contents</u>	<u>Explanation</u>	<u>Action</u>
102055	A line is about to be printed.	Turn punch unit OFF. Press RUN.
102056	A line has been printed.	Turn punch unit ON. Press RUN.

Operating Instructions

The following procedures indicate the sequence of steps for loading and execution of the Basic Control System Loader:

- A. Set Teleprinter to LINE and check that all equipment to be used is operable.
- B. Load the Basic Control System tape using the Basic Binary Loader:
 1. Place the Basic Control System tape in the device serving as the Standard Input unit (e.g., Punched Tape Reader).
 2. Set Switch Register to starting address of Basic Binary Loader (e.g., 007700 for 4K memory, 017700 for 8K memory).
 3. Press LOAD ADDRESS.
 4. Set LOADER switch to ENABLED.
 5. Press PRESET.
 6. Press RUN.
 7. When the computer halts and indicates that the BCS tape is loaded (T-Register contains 102077), set the LOADER switch to PROTECTED.
- C. Set Switch Register to 000002, press LOAD ADDRESS, and set Switch Register to 000000.
- D. Establish Loader parameters:
 1. Set Switch 15 to 1 if no Memory Allocation Listing is desired during first load operation.
 2. Set Switch 14 to 1 if an absolute binary tape of the programs is to be punched. (Turn on punch device if this option is selected.)

E. Place relocatable object tape in device serving as Standard Input unit.

F. Press RUN.

During the operation of the Basic Control System Loader, the following halts may occur:

<u>Teleprinter Message</u>	<u>Explanation</u>	<u>Action</u>
LOAD (T-Register contains 102001.)	End-of-tape condition on Standard Input device.	<ol style="list-style-type: none">1. To load next tape, set Switches 2-0 to 0g. If no Memory Allocation Listing of next tape is desired, set Switch 15 to 1. Press RUN to continue loading.2. To indicate that all programs are loaded and to proceed to the end-of-loading phase, set Switches 2-0 to 1g. Press RUN.†3. To terminate loading operation, set Switches 2-0 to 2g. Press RUN. (This forces execution even though undefined external references have not been matched.)

†A list of any undefined external symbols is typed following a Switch Register reply of 1g or 4g to the "LOAD" message. The message "LOAD" is then repeated. The programs containing the matching entry points should be loaded. Loading of user programs from Standard Input must be completed before loading of routines from Program Library.

Teleprinter
Message

Explanation

Action

		4. To load from Program Library, set Switches 2-0 to 48. If no Memory Allocation Listing of library routines is desired, set Switch 15 to 1. Press RUN to continue loading. When all library routines are loaded, the Loader proceeds directly to the end-of-loading phase.
*LST	The Loader is ready to print the LST, common bounds, and linkage area bounds.	If a list of these items is not desired, set Switch 15 to 1. Press RUN.
*RUN	All programs are loaded and ready for execution.	Check that all I/O devices are ready for operation. Press RUN.
*END	The absolute binary output has been selected and the punched tape is complete.	To execute the program: 1) Load binary tape using Basic Binary Loader as in B. 2) Set Switch Register to 000002. Press LOAD ADDRESS. 3) Press RUN.
*L01	Checksum error: The checksum read on the last record does not agree with the checksum calculated by the Loader.	To re-read record, reposition tape to beginning of record and press RUN.

Teleprinter
Message

Explanation

Action

- | | | |
|------|---|--|
| *L02 | Illegal record: The last record read was not recognized as one of the five types accepted by the Loader. | To re-read record, reposition tape to beginning of record and press RUN. |
| *L03 | Memory overflow: The length of the main or Base Page portion of the program or the common block exceeds the bounds of available memory. | Irrecoverable error, program must be revised. |
| *L04 | Linkage area overflow: Linkage words supplied by the Loader for references between pages exceed the size of available base page memory. | If program consists of several subprograms, altering the sequence in which the subprograms are loaded may reduce the number of linkage words. Otherwise, irrecoverable error, program must be revised. |
| *L05 | Loader symbol table overflow: The number of EXT/ENT symbols exceed available memory. | Irrecoverable error, program must be revised. |
| *L06 | Common block error: The length of the common block in the current program is greater than the length of the first common block allocated. | Revise sequence in which subprograms are loaded and reload program. Otherwise, revise program. |
| *L07 | Duplicate entry points: An entry point in the current program matches a previously declared entry point. | Irrecoverable error, program must be revised. |

Teleprinter
Message

Explanation

Action

- | | | |
|------|--|--|
| *L08 | No transfer address:
The initial starting lo-
cation (e.g., END
statement operand)
was not present in any
of the programs which
were loaded. | To enter the starting
address, set the abso-
lute value in the Switch
Register, press LOAD
A, and press RUN. |
| *L09 | Record out of se-
quence: ANAM record
was encountered be-
fore the previous pro-
gram was terminated
with an END record. | Irrecoverable error,
program must be re-
vised. |

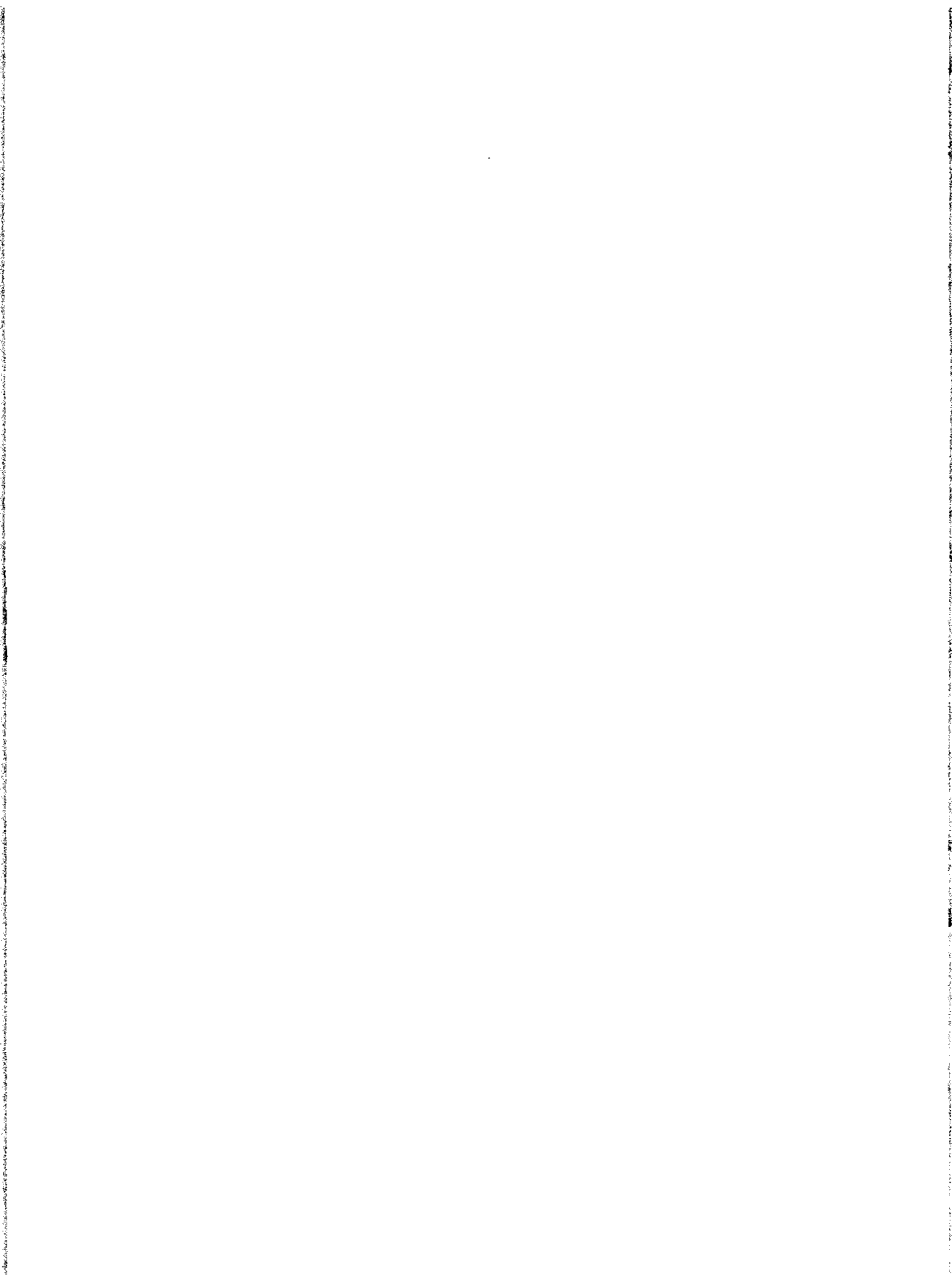
If the absolute binary output option is selected, the following halts may occur:

T-Register
Contents

Explanation

Action

- | | | |
|--------|--|--|
| 102066 | Tape supply low on
2753A Tape Punch
which is producing ab-
solute binary output.
Trailer follows last
valid output. | Place new reel of tape
in unit. Press RUN.
Leader is produced. |
| 102055 | A line is about to be
printed on Teleprinter
output device. (See
Separation of List and
Binary Output.) | Turn punch unit off.
Press RUN. |
| 102056 | A line has been printed
while Teleprinter
punch unit off. (See
Separation of List and
Binary Output.) | Turn punch unit on.
Press RUN. |



4.1 GENERAL DESCRIPTION

An I/O driver, operating in the BCS environment, is responsible for controlling all data transfer between an I/O device and the cpu. It operates under control from the program IOC. Its operating parameters are the user I/O request and the information contained in the device associated Equipment Table entry.

4.2 STRUCTURE

An I/O driver is a relocatable program segmented into two closed subroutines, termed the "initiator" and "continuator" sections. The entry point names for these two sections must be "D.nn" and "I.nn", respectively. The numeric value "nn" in the names is the Equipment Type Code assigned to the device. For example, D.00 and I.00 are the entry points for the Teletype driver; "00" is the Equipment Type Code assigned to a Teletype.

NAM DRIVER D.nn

D.nn

Initiator Section

I.nn

Continuator Section

4.2.1 Initiator Section

This section is called directly from IOC with calling parameters including the address of the second word of the user I/O request and the address of the EQT entry for the referenced device. IOC sets these parameters in A and B and performs a JSB to the entry point "D.nn". Return to IOC from this section must be indirectly through D.nn.

On entry to D.nn,

- (A) = Address of word 1 of 4-word EQT entry
- (B) = Address of word 2 of I/O request

The initiator section of any driver must perform the functions described below.

- 1) Reject the IOC request and return to IOC (see step 6) if any of the following conditions exist:
 - a. the driver is busy operating another device
 - b. the referenced device is busy or inoperable
 - c. the user request code or other parameters illegal for the device
 - d. a DMA channel is not available and DMA is required for data transfer.
- 2) Extract the parameters from the user I/O request and save them within the driver storage.
- 3) Configure all I/O instructions in the driver to include the channel number for the reference device.
- 4) Indicate equipment in operation:
 - a. set the "a" field in the EQT entry to 2 (busy) for the device called
 - b. set an internal driver "busy" flag for the driver
 - c. set a "busy" flag in IOC if a DMA channel is used

(To set a DMA flag in IOC:

Within the IOC program the two entry points DMAC 1, DMAC 2 contain the DMA channel locations (6 and 7 or 7 and 6). The signbit of the channel used must be set to 1 to indicate that the channel is busy.)

- 5) Initialize operating conditions and activate the device.
- 6) Return to IOC with the A and B registers set to indicate initiation or rejection and the cause of the reject:

(A) = 0, operation initiated
 = 1, operation rejected - reason in B-register

(B) = 1000000, the device is busy or inoperable, or the driver is busy
 = 0000001, a DMA channel is required but no channel is available
 = 0000000, the request code or sub-function is not legal for the device

4.2.2 Continuator Section

This section is entered by device interrupt to continue or complete an operation. It may also be called from the Initiator Section to begin an operation. The entry point to this section is I.nn. There are no parameters on entry.

The continuator section of any driver must perform the functions described below.

- 1) Save all registers which will be used by the continuator section.
- 2) Perform the input or output of the next data item. If the transfer is not completed, restore the "saved" register and return control to the program (see steps 5 and 6).

NOTE: A driver for a device which inputs or outputs data independent of program control such as DMA would not include step 2. The device is turned "on" by the initiator section (step 5) and the data transfer is immediately accomplished. The continuator section for such drivers merely completes the input or output operation.

- 3) When data transfer is completed (end-of-operation) or if a device malfunction is detected, set the following information in the EQT entry:

The number of words or characters transferred (corresponding to the request) is set as a positive value in word 3. Bit 15 of word 3 is set to 0 or 1 to indicate the mode of transfer.

The device status, actual or simulated, is set in bits 07-00 of word 2 and the "a" field (bits 15-14) in word 2 set to:

0 - device available (not busy)

1 - device available; the operation is complete but an error has been detected

Bits 13-08 of word 2 must not be altered.

- 4) Clear all "busy" indicators. Clear the driver busy flag. If a DMA channel was used clear the flag in IOC.
- 5) Restore all registers saved at the entry.
- 6) Return indirectly through the entry point I.nn, with the following exception:

If end-of-operation and the operation completed was an output or Function request, return must be made to the entry point ".BUFR" in IOC. This enables the Buffered version of IOC to perform the automatic output buffering function. The standard version of IOC at this entry point just performs a normal return to the point of interruption. The calling sequence to .BUFR is:

	EXT	.BUFR
(P)	JSB	.BUFR
(P+1)	NOP	(holds return address from I.nn)
(P+2)	NOP	(holds EQT entry address)

The Prepare Control System (PCS) program processes relocatable modules of the Basic Control System and produces an absolute version designed to work on a specific hardware configuration. It creates operating units of the Input/Output Control subroutine (.IOC.), the equipment driver subroutines, and the Relocating Loader. It also establishes the contents of certain locations used in interrupt handling. Options are available to define the equipment driver modules and other BCS system subroutines as relocatable programs to be loaded with the user's object program.

The Prepare Control System is an absolute program which is loaded by the Basic Binary Loader. It operates on a minimum configuration of 4K memory and a 2752A Teleprinter. However, if a 2737A Punched Tape Reader and a 2753A Tape Punch are available, the Prepare Control System will utilize these devices; PCS requests their assignment during the Initialization phase.

After the Initialization phase is completed, each module of BCS is loaded and processed by PCS. The order in which the modules are processed is not significant except that the BCS Loader must be the last module loaded. Two modules, the Input/Output Control subroutine and the Loader, require that parameters be entered via the Keyboard Input unit after being loaded.

5.1 INITIALIZATION PHASE

During the Initialization phase, the System requests that the operator provide the channel assignments of the Punched Tape Reader and the Tape Punch if available. This is followed by a request for the first and last words of available memory. The first word is the location in the Base Page following the locations required for interrupt processing (the interrupt locations and the locations containing the addresses of the Interrupt Processors). This location defines the start of the BCS System Linkage area. The last word of available memory is usually the location prior to the protected area (e.g., 7677 for 4K memory, 17677 for 8K memory).

Example:

HS INP?	message
10	reply
HS PUN?	message
11	reply
FWA MEM?	message
30	reply
LWA MEM?	message
17677	reply
.	
.	
.	

5.2 LOADING OF BCS MODULES

After the Initialization phase is completed, the System types the message "LOAD". The modules of BCS are then loaded using the Punched Tape Reader, if available, or the Teleprinter reader. The modules may include .IOC., the equipment drivers, and the Relocating Loader. They may be loaded in any order provided that the Relocating Loader is last. The message is repeated after each module is loaded until the Loader has been processed. Diagnostics are printed if certain error conditions occur during the loading. The absolute lower and upper bounds of each program within BCS are listed after the program is loaded.

The format is as follows:

<program name>

lllll uuuuu

Equipment driver subroutines and interrupt processing sections which are to be used in relocatable form are identified during PCS processing but are not loaded. At the completion of the processing, PCS requests the missing subroutines. The proper response identifies each as relocatable.

5.3 INPUT/OUTPUT EQUIPMENT PARAMETERS

After the Input/Output Control module is loaded, PCS requests the information needed to construct the Equipment Table (EQT) and Standard Equipment Table (SQT).†

Equipment Table Statements

PCS first types the messages "TABLE ENTRY" and "EQT". The operator responds by supplying the Equipment Table entries in the following format:

nn, D.ee [,D] [,Uu]

- nn The channel number (select code) for the device. For a device connected to two or more channels, nn is the lower numbered channel.
- D.ee The Basic Control System symbolic name for the related equipment driver subroutine. ee is the equipment type code used by BCS. Driver names are as follows:
- D.00 - 2752A Teleprinter
 - D.01 - 2737A Punched Tape Reader
 - D.02 - 2753A Tape Punch
 - D.20 - Kennedy 1406 Incremental Tape Transport
 - D.21 - 2020A Magnetic Tape Unit
 - D.22 - 3030A Magnetic Tape Unit
 - D.40 - Data Source Interface
 - D.41 - Integrating Digital Voltmeter
 - D.42 - Guarded Crossbar Scanner
 - D.43 - Time Base Generator
- D A Direct Memory Access channel is required to operate the device.
- Uu The physical unit number u(0-7) for addressing the device if it is attached to a multi-unit controller.

The same response is used regardless of whether the related subroutinedriver is to be relocatable or absolute (part of BCS). If the driver is not encountered during processing, PCS prints the following:

I/O DRIVER?
D.EE

†See Appendices B and C for description of EQT and SQT.

A response of "!" indicates that the driver is to be in relocatable form. (Any other response at this time is an error.)

The order in which the EQT statements are submitted defines the position of the entry in the Equipment Table. It also establishes the unit-reference number that the programmer uses in writing input/output requests to .IOC. The first statement entered describes the unit which is to be referenced as number 7g; the second statement, number 10g; the third statement, number 11g; etc. Numbers 1 through 6 are reserved for Standard unit definition in the Standard Equipment Table.

The statement "/E" is entered to terminate the EQT input.

Example:

		<u>Unit-Reference Number</u>
*TABLE ENTRY	Message	
EQT?	Message	
10.D.01	Statement	7
11.D.02	Statement	10
12.D.00	Statement	11
/E	Terminator	

Standard Equipment Table Statements

In constructing the Standard Equipment Table, PCS types a mnemonic for the Standard unit and waits for the reply. The reply consists of the unit-reference number for a device previously described in the Equipment Table.

Example:

SQT?	message
-KYBD?	message to assign Keyboard Input
11	reply: unit-reference number for Teleprinter
-TTY?	message to assign Teleprinter Output
11	reply: unit-reference number for Teleprinter
-LIB?	message to assign Program Library

7	reply: unit-reference number for Punched Tape Reader
-PUNCH?	message to assign Punch Output
10	reply: unit-reference number for Tape Punch
-INPUT?	message to assign Input
7	reply: unit-reference number for Punched Tape Reader
-LIST?	message to assign List Output
11	reply: unit-reference number for Teleprinter

Direct Memory Access Statement

After the equipment tables are completed, PCS requests information about the availability of DMA channels to be controlled by the Input/Output Control and equipment driver subroutines. PCS types the message "DMA?" and the operator responds with the available DMA channel numbers. The format of the reply is:

$$c_1 [, c_2]$$

c_1 is 6 if one channel is available

c_2 is 7 if the second channel is available

If no DMA channel is available, the reply is 0 (zero).

Example:

DMA?	message
6, 7	reply for two channels

If the reply contains any characters other than 6 or 7 it is considered to be in error and a diagnostic is issued.

5.4 INTERRUPT LINKAGE PARAMETERS

After the Relocating Loader is loaded, PCS requests the parameters needed to set the Interrupt Linkage for Input/Output processing. The information required for each device includes:

The interrupt location within the Reserved Location area in low core.

The entry point name of the interrupt processing section in the equipment driver subroutine for the device.

The address of the word in the Base Page which is to contain the 15-bit absolute address of this entry point name.

The same response is used regardless of whether the subroutine driver is to be relocatable or absolute (part of BCS). If the entry point was not encountered during processing, PCS prints the following:

*UN NAME

A response of "" indicates that the driver is to be in relocatable form. (Any other response at this time redefines the linkage.)

Given this information, PCS sets in the interrupt location a Jump Subroutine (Indirect) to the word holding the absolute address for the entry point of the Interrupt Processor.

<u>Location</u>	<u>Content</u>
10	JSB 20B,I
.	.
.	.
.	.
20	DEF I.01

10 is the interrupt location

20 holds the address of the entry point, I.01, of the Interrupt Processor.

PCS types the message "INTERRUPT LINKAGE?" The operator responds with a message in the following format:

$a_1, a_2, I.ee$

- a_1 The address in low core of the interrupt location for the device (channel).
- a_2 The address in the Base Page of the word to contain the absolute address of the Interrupt Processor entry point.
- $I.ee$ The entry point name of the Interrupt Processor section of the equipment driver subroutine. ee is the equipment type code used by BCS. Entry point names are as follows:
- $I.00$ - 2752A Teleprinter
 - $I.01$ - 2737A Punched Tape Reader
 - $I.02$ - 2753A Tape Punch
 - $I.20$ - Kennedy 1406 Incremental Tape Transport
 - $I.21$ and $C.21^\dagger$ - 2020A Magnetic Tape Unit
 - $I.22$ and $C.22^\dagger$ - 3030A Magnetic Tape Unit
 - $I.43$ - Time Base Generator

The statement "/E" is entered to terminate the Interrupt Linkage parameter input.

Example:

INTERRUPT LINKAGE?	message
10,20,I.01	reply: The Punched Tape Reader uses interrupt location 10. The absolute address for entry point I.01 is location 20 in the Base Page.
11,21,I.02	reply: The Tape Punch uses interrupt location 11. The address of I.02 is at location 21.

† Both the magnetic tape systems are connected to two channels; the lower numbered channel transfers data (D.21, D.22); the higher numbered channel transfers commands (C.21, C.22).

12,22,1.00

13,22,1.00

reply: The Teleprinter, which requires two channels, uses interrupt locations 12 and 13. The Interrupt Processor entry point address is stored at location 22.

/E

Terminates linkage parameters.

The response to the "INTERRUPT LINKAGE?" message may have the following form if a constant, for example a halt, is to be set in the interrupt location.

a, c

a The address in low core of the interrupt location for the device (channel).

c The constant in octal form that is to be stored at location a.

Example:

INTERRUPT LINKAGE?

message

27,102027

reply: A halt executed when interrupt occurs on channel 27.

26,0

reply: A NOP is executed when interrupt occurs on channel 26; the program resumes normal execution.

5.5 PROCESSING COMPLETION

When the Interrupt Linkage parameters have been supplied, PCS performs the following functions:

1. Prints the message "*UNDEFINED SYMBOL" followed by the entry point names of all system subroutines which have been referenced as externals but not loaded. At this point, PCS may be rerun and the missing subroutines loaded or, the symbols may be added to the Relocating Loader's Loader Symbol Table. Undefined symbols are assigned a value of 77777 for an absolute address.

2. Completes the construction of the Loader Symbol Table.
3. Sets the Memory Table (symbolic location .MEM.) in the Relocating Loader to reflect the final bounds of available memory.

Following this, PCS prints a list of all Basic Control System entry points and the bounds of the System Linkage area in the Base Page.

Example:

```

.SQT.      17472
.EQT.      17500
.IOC.      17515
DMAC1      17676
DMAC2      17677
IOERR      17656
XSQT       17674
XEQT       17675
D.00       16745
I.00       17107
D.01       16406
I.01       16521
D.02       16115
I.02       16226
.LDR.      15413
HALT       16110
.MEM.      16110
LST        14102

```

```

*SYSTEM LINK
00030 00071

```

The final step in PCS processing is the punching of an absolute binary tape of the configured Basic Control System. This tape can be loaded by the Basic Binary Loader. When the tape has been punched, BCS types the message "*BCS ABSOLUTE OUTPUT". At the completion of the PCS run, the message "**END" is typed. The tape is punched using the Tape Punch unit if available, or the Teleprinter punch.

5.6 OPERATING PROCEDURES

The following procedures indicate the sequence of steps for loading and execution of the Prepare Control System.

- A. Set Teleprinter to LINE and check that all equipment to be used is operable.
- B. Load the Prepare Control System tape using the Basic Binary Loader.
 1. Place the Prepare Control System tape in the device serving as the Standard Input unit (e.g., Punched Tape Reader).
 2. Set Switch Register to starting address of Basic Binary Loader (e.g., 007700 for 4K memory, 017700 for 8K memory).
 3. Press LOAD ADDRESS.
 4. Set LOADER switch to ENABLED.
 5. Press PRESET.
 6. Press RUN.
 7. When the computer halts and indicates that the PCS tape is loaded (T-Register contains 102077) set the Loader Switch to Protected.
- C. Set Switch Register to 002000, press LOAD ADDRESS.
- D. Set Switches 5-0 to the value of the channel number of the Teleprinter. (On a two channel teleprinter use the lower numbered channel.)
- E. Press RUN.

The initialization Phase is executed. During this phase the following messages may occur:

<u>Teleprinter Message</u>	<u>Explanation</u>	<u>Action</u>
HS INP ?	Request for Punched Tape Reader channel assignment.	Type channel number. If Punched Tape Reader not available, type 0. †
HS PUN ?	Request for Tape Punch channel assignment.	Type channel number. If Tape Punch not available, type 0.
FWA MEM ?	Request for first word of available memory.	Type address of word in Base Page following the locations required for interrupt processing.
LWA MEM ?	Request for last word of available memory.	Type address of word preceding protected area.
*ERROR	A non-numeric or illegal character has been entered as a reply.	Type the correct value.

Following the completion of the Initialization Phase the relocatable object tapes of the Basic Control System are to be loaded. Only those modules which are to be included in the absolute tape are loaded; modules which are to be loaded with the user's object program are not submitted. The modules may include .IOC., the equipment drivers, and the Relocating Loader; they may be loaded in any order provided that the Relocating Loader is last. During this phase, the following halts may occur:

† All replies from the keyboard must be terminated by an end-of-statement mark which consists of a carriage return, (CR), and a line feed, (LF). If an error is made in typing a reply, type (RUBOUT) (CR) (LF) and repeat the reply.

Teleprinter
Message

Explanation

Action

*LOAD	PCS is requesting the first or the next BCS module.	Place BCS tape in Punched Tape Reader if available, or Teleprinter reader. Press RUN.
*L01	Check sum error	To re-read record, reposition tape to beginning of record and press RUN.
*L02	Illegal record: the last record read was not recognized as a valid relocatable record type.	To re-read record, reposition tape to beginning of record and press RUN.
*L03	Memory overflow: the length of BCS exceeds available memory.	Irrecoverable error.
*L04	System linkage area overflow in Base Page.	Irrecoverable error.
*L05	Symbol table for BCS symbols exceeds available memory.	Irrecoverable error.
*L06	PCS interprets the program length of BCS to be zero.	Irrecoverable error.
*L07	Duplicate entry points within BCS.	Irrecoverable error.
*EOT	End-of-Tape	Place next tape in read unit and press RUN to continue loading.

When the .IOC. module is loaded, PCS requests the EQT and SQT parameters. PCS halts after typing the messages "*TABLE ENTRY? EQT?". If the Teleprinter serves both as the reader and keyboard unit, turn reader off, press RUN. Begin typing response to message. (Turn reader on after all replies have been typed.)

Teleprinter
Message

Explanation

Action

*TABLE	Request for EQT entry information.	Press Run and for each I/O device, type: nn, D. ee, [, D] [, Uu] nn - channel number D. ee - driver name: ee = 00 Teleprinter = 01 Punched Tape Reader = 02 Tape Punch = 20 Kennedy 1406 Incremental Tape Transport = 21 2020A Magnetic Tape Unit = 22 3030 Magnetic Tape Unit = 40 Data Source Interface = 41 Integrating Digital Voltmeter = 42 Guarded Crossbar Scanner = 43 Time Base Generator D - device uses DMA channel Uu - physical unit number (0-7) if attached to multi-unit controller To terminate EQT input, type /E.
*ERROR	A non-numeric value has been typed for nn, ee, or u.	Retype the entire correct entry.
SQT? -KYBD?	Request for EQT unit-reference number of unit serving as Keyboard Input.	Type number.
-TTY?	Request for EQT unit-reference number of unit serving as Teleprinter Output.	Type number.

<u>Teleprinter Message</u>	<u>Explanation</u>	<u>Action</u>
-LIB ?	Request for EQT unit-reference number of unit serving as Program Library.	Type number.
-PUNCH ?	Request for EQT unit-reference number of unit serving as Punch Output.	Type number.
-INPUT ?	Request for EQT unit-reference number of unit serving as Input.	Type number.
-LIST ?	Request for EQT unit-reference number of unit serving as List Output.	Type number.
DMA ?	Request for DMA channel numbers.	If one DMA channel, type 6. If two DMA channels, type 6, 7. If no DMA channels, type 0.
*ERROR	A non-numeric parameter or a parameter not equal to 6 or 7 has been entered.	Re-type correct parameter.

After the Relocating Loader is loaded, PCS requests the information needed to set the interrupt linkage for input/output processing. PCS halts after typing the message "INTERRUPT LINKAGE?". If the Teleprinter is serving both as the reader and the keyboard unit, turn reader off and press Run. Begin typing response to message.

Teleprinter
Message

Explanation

Action

INTERRUPT Request for interrupt
LINKAGE? information.

Press Run and for each I/O
device, type:

a_1, a_2, I, ee

a_1 - interrupt location address

a_2 - location containing absolute address of Interrupt
Processor entry point

I, ee entry point name:

$ee = 00$ Teleprinter
= 01 Punched Tape Reader
= 02 Tape Punch
= 20 Kennedy 1406 Incremental Tape Transport
= 21 (and C.21) 2020A Magnetic Tape System
= 22 (and C.22) 3030A Magnetic Tape System
= 43 Time Base Generator

If a constant is to be set into
the interrupt location, type:

a, c

a - interrupt location address
 c - 1 to 6 digit octal constant
to be stored at a .

Constants should be entered
for the following instrument
drivers:

Data Source Interface (D. 40):
1067 sc (CLC sc, 4)
Integrating Digital Voltmeter
(D. 40): \emptyset (NOP)
Guarded Crossbar Scanner
(D. 42): \emptyset (NOP)

To terminate linkage input,
type /E.

Teleprinter
Message

Explanation

Action

- | | | |
|----------------|--|---|
| *ERROR | A non-numeric value has been typed for a ₁ , a ₂ , a or c. | Retype the entire correct entry. |
| *UNNAME | The name I. ee is not defined as an entry point in any I/O driver previously loaded. | <ol style="list-style-type: none">1) If the driver name was typed incorrectly, retype the entire correct entry.2) If related driver is to be loaded with user's program at object program load time, type an exclamation mark (!). The name is added to the Loader's LST.3) If the driver should have been loaded, rerun PCS. |

When the Interrupt Linkage parameters have been supplied, PCS performs the following functions:

1. Prints the message **"*UNDEFINED SYMBOL"** followed by the entry point names of all system subroutines which have been referenced as externals but not loaded. At this point, PCS may be rerun and the missing subroutines loaded or, the symbols may be added to the Relocating Loader's Loader Symbol Table. Undefined symbols are assigned a value of 77777 for an absolute address.
2. Completes the construction of the Loader Symbol Table.
3. Sets the Memory Table (symbolic location .MEM.) in the Relocating Loader to reflect the final bounds of available memory.

Following this, PCS prints a list of all Basic Control System entry points and the bounds of the System Linkage area in the Base Page.

Teleprinter
Message

Explanation

Action

*UNDE-
FINED
SYMBOL
<symbol>

An entry point in a BCS module cannot be located.

1) To enter the symbol in the Loader Symbol Table, press RUN.

2) If the subroutine should have been loaded, rerun PCS.

I/O DRIVER?
D. ee

A driver has been named in the EQT parameter entry, but has not been loaded.

1) If the driver is to be loaded with user's program at object program load time, type an exclamation mark (!). The name is added to the Loader's LST.

2) If the driver should have been loaded (or if a character other than ! is typed), rerun PCS.

*BCS
ABSOLUTE

PCS is ready to punch absolute output tape.

Turn on punch unit and press RUN.

When the binary tape is punched the following halts may occur:

T-Register
Contents

Explanation

Action

102077

BCS tape is punched.

To produce additional copies, set Switch 15 to 1 and press RUN.

102066

Tape supply low on 2753A Tape Punch.

Place a new reel of tape in Tape Punch and press RUN to continue.

5.7 EXAMPLE

The teleprinter listing shown below was output from a Prepare Control System operation that configured a BCS tape for an HP 2116 system which included a Teleprinter, a Tape Punch, a Punched Tape Reader, and a 2020A Magnetic Tape Unit.

The PCS tape was loaded; 2000_g was set in the P-Register; the Switch Register was set to the channel number of the teleprinter, in this case the lower channel number (13_g). The RUN button was pressed and the Teleprinter printed:

HS INP?
10
HS PUN?
11

Initialization Phase

FWA MEM?
23
LWA MEM?
17677

* LOAD

D•21
16231 17677

Magnetic Tape

* LOAD

D•01
15672 16230

Punch Tape Reader

* LOAD

Load the BCS
Drivers

D•02
15362 15671

Tape Punch

* LOAD

D•00
14606 15361

Teleprinter

* LOAD

IOC

14367 14605

Load IOC

* TABLE ENTRY

EQT?

10,D.21

12,D.01

13,D.02

14,D.00

/E

Enter the EQT Table

SQT?

-KYBD?

12

-TTY?

12

-LIB?

7

-PUNCH?

11

-INPUT?

10

-LIST?

12

Enter the SQT Table

DMA?

0

* LOAD

LOADR

12170 14337

Load the Reloactable Loader

INTERRUPT LINKAGE ?

```
10,16,I.21
11,17,C.21
12,20,I.01
13,21,I.02
14,22,I.00
15,22,I.00
/E
```

Enter the Interrupt Linkage

```
.SQT. 14340
.EQT. 14346
D.21 16231
I.21 17216
C.21 17130
D.01 15672
I.01 16007
D.02 15362
I.02 15476
.BUFR 14535
D.00 14606
I.00 14755
.IOC. 14367
DMAC1 14604
DMAC2 14605
IOERR 14563
XSQT 14602
XEQT 14603
.LDR. 13624
HALT 14333
.MEM. 14333
LST 12214
```

BCS entry points

```
*SYSTEM LINK
00023 00166
```

Call to turn on the tape punch for absolute binary tape of the configured BCS.

```
*BCS ABSOLUTE OUTPUT
*END
```

The Debugging routine provides the following facilities to aid in program testing:

- Print (dump) selected areas of memory in octal or ASCII format
- Trace portions of the program during execution
- Modify the contents of selected areas in memory
- Modify simulated computer registers
- Instruction and operand breakpoint halts
- Initiate execution at any point in program
- Debugging routine restart
- Specifying relocatable program base

The Debugging routine supervises the operation of a program in the check-out (debugging) phase through the use of an interpretive mode of execution with simulated A, B, E overflow and P registers.

The Debugging routine is a relocatable program. It is loaded into memory after the user's relocatable programs and before the library subroutines are loaded. The Debugging routine makes use of the input/output control subroutine, IOC.

6.1 OPERATOR COMMUNICATION

All communication between the Debugging routine and the user formed by the Standard Keyboard Input and Standard Teleprinter Output units which are normally assigned to a Teleprinter.

After the program is loaded, the Debugging routine pauses to allow the first type-in. The operator then types one or more control statements to direct the operation of the Debugging routine. Each statement must be terminated by an end-of-statement mark which consists of a carriage return, CR , and a line feed LF . The last statement of the set must be a Run statement.

When an operation requested by a control statement is completed, a pause occurs (except for the Trace operation). The operator may then continue by typing a Run statement, or he may enter new control statements. To regain control at any

other time, the operator must use Switch 15. Caution must be used, however, when input/output operations are in progress; setting the switch causes a message to be typed. This action may disrupt any incomplete I/O operation.

6.2 CONTROL STATEMENTS

The basic format of the control statement is a single alphabetic character representing the requested operation followed by a parameter list containing the arguments for the operation separated by commas. The statement is of variable length and is terminated by **(CR)** **(LF)**. The numeric fields in the parameter list must be in octal; leading zeros may be omitted.

Program Relocation Base

M, a

This statement defines the program relocation base, a, as the absolute origin in memory of the user's relocatable program. This address may be obtained from the listing produced by the Relocating Loader during loading. If not specified, a value of zero is assumed. The value is added to all address parameters entered by the operator.

Specification of this value allows subsequent reference in the control statements to addresses as shown on the program listing produced by the Assembler or the FORTRAN compiler. If this control statement is not used, program address parameters for other control statements must be absolute.

Example:

M, 20000

Set Memory

S, a, v₁, v₂, ..., v_n

The above statement allows the user to set one or more values into locations defined by the first address, a. The value specified for v₁ is stored in location a; the value for v₂, in location a + 1; and so forth. To specify that an existing value in memory is to remain unchanged, two consecutive commas are used in the control statement. Any number of values may be entered via one control statement provided the length of the statement does not exceed 72 characters.

Example:

S, 5, 062006
S, 30, 136100, 026040
S, 40, 136101, 026050

Set Register

W, r, v

This statement sets the value, v, into register, r, where the register is defined as follows:

r = A, A-Register
= B, B-Register
= E, E-Register
= O, Overflow

Since the Debugging routine simulates the register, the results of a Set Register operation are not reflected on the computer front panel.

Examples:

W, B, 2
W, A, 102000
W, E, 1

Dump Memory

D, A, a₁, a₂
D, B, a₁, a₂

The second parameter indicates the format of the print-out: A specifies ASCII, B specifies octal (See Output Formats). The address a₁ designates the location of the word or the first of a series of words that is to be dumped. If the second address, a₂, is greater than a₁, a block of memory, a₁ through a₂, is printed. If a₂ is the same as a₁, only one location is printed.

After the data is printed, the Debugging routine waits for the operator to enter another control statement.

Example:

D, A, 430, 477

Breakpoint Halt

B, I, a
B, O, a

The first form specifies the address, a, of an instruction breakpoint. Before the instruction at address a is executed, the Debugging routine writes a standard breakpoint message (See Output Formats).

The second form specifies the address, a, of an operand breakpoint. When the Debugging routine detects an effective operand address equal to the value of a, it writes a standard breakpoint message. The operand breakpoint occurs before the memory reference is completed and the register contents in the message are the contents during the instruction execution and not at completion.

After the breakpoint message is transmitted, the Debugging routine waits for the user to enter another control statement.

One or both types of breakpoint halts may be selected. Once selected, a breakpoint address remains in effect until a new address is selected, until a Restart statement is entered, or until the selection is terminated by the statements:

B, I, Ø or B, O, Ø

T, a₁ [, a₂]

Trace

When the Trace operation is specified, the execution of the instruction located at address A₁, or the execution of every instruction within the area a₁ through a₂, causes the printing of a standard breakpoint message (See Output Formats). The printing occurs before each instruction is executed. Each time the a₁ - a₂ area is reached, the printing resumes; no pause occurs on completion as in the other Debugging routine operations.

The area to be traced must not contain calls to the input/output control routine, IOC. The Trace operation uses IOC to print the breakpoint message. An attempt to trace I/O operations will result in I/O errors.

The trace of the area remains in effect until a new area is selected or until the selection is terminated by the statement:

T, Ø

To enter a new Trace control statement while the program is in operation, Switch 15 must be used.

Run

R [, a]

This statement is used to initiate or continue the execution of the program being debugged. As a result of using Switch 15, or following the printing of a standard breakpoint message (with the exception of those produced by Trace), the Debugging program pauses. The operator may then type a Run statement, or one or more control statements followed by a Run statement to resume program execution. If the letter "R" only is entered, execution starts with the next sequential instruction in the user's program. To start at another location, the operator enters the address, a.

Restart

A

This statement, consisting of the letter "A" is used to abort the current operation and restart. This results in all debugging routine and input/output operations in progress being cleared.

6.3 CONTROL STATEMENT ERROR

If an incorrect control statement is entered, the following message is typed:

"ENTRY ERROR"

This indicates that the character representing the operation is invalid, or that an illegal parameter has been typed. To recover, type in the correct control statement.

6.4 HALT

Any Halt operations coded within the user's program are interpreted by the Debugging routine and result in a typeout consisting of the letter "H" followed by the standard breakpoint message. The operator may then type in one or more control statements or may reinitiate program execution (with the R control statement).

6.5 INDIRECT LOOP

The Debugging routine maintains a count of levels when indirect addressing is detected. When ten consecutive levels of indirect addressing have occurred, an indirect address loop is assumed and the following is typed out:

"INDIRECT LOOP"

L <standard breakpoint message>

6.6 OUTPUT FORMATS

The Debugging routine operations may produce either of two printed outputs: the standard breakpoint message and the memory dump.

Standard Breakpoint Message

Each output line from operations which produce the standard breakpoint message has the following format:

<id> P = v₁ I = v₂ A = v₃ B = v₄ E = v₅ O = v₆ MA = v₇ MC = v₈

The <id> is a letter identifying the operation producing the output:

- id = I, Instruction breakpoint
- = O, Operand breakpoint
- = T, Trace
- = S, Switch 15 set up
- = L, Indirect Loop
- = H, Halt in object program

The v's are octal values of registers and memory locations as follows:

- P - P-Register (instruction address)
- I - Instruction (contents)
- A - A-Register
- B - B-Register
- E - E-Register
- O - Overflow
- MA - Effective operand address of a memory reference instruction
- MC - Contents of effective address of a memory reference instruction

Dump

The Dump output record format consists of the contents up to 8 consecutive words preceded by the address of the first word:

	<u>addr.</u>	<u>word₁</u>	<u>word₂</u> ...	<u>word₈</u>
Octal:	aaaaa	000000	000000 ...	000000
ASCII:	aaaaa	cc	cc	cc

Octal words consist of 6 octal digits; ASCII words are listed as two ASCII characters. The contents of eight or more consecutive words are not written or they are the same as the last word of the previous record. Instead, a record containing only an asterisk is produced.

6.7 OPERATING PROCEDURES

The following procedures indicate the sequence of steps for use of the Debugging routine.

- A. Set the Teleprinter to LINE and check that all equipment to be used is operable.
- B. Load Basic Control System using the Basic Binary Loader.
- C. Set Switch Register to 000002, press LOAD ADDRESS, and set Switch Register to 000000.
- D. Establish Relocating Loader parameters. (If relocation base is to be entered during operation of the Debugging routine, the address must be obtained during loading by setting Switch 15 to 0 (down).)
- E. Load relocatable object programs.
- F. Load Debugging program (treated as a relocatable program). †
- G. Load Program Library routines.

† The Debugging routine need not be loaded as the last relocatable program. If loaded in any other order, however, the absolute address assigned to the symbolic location DEBUG must be entered manually as the starting address for the program.

H. Press RUN.

I. The program pauses to allow the operator to type in the control statements.

J. The program may be restarted at any point by entering the absolute address assigned to the symbolic location DEBR5 into the P-Register, and pressing RUN.

6.8 EXAMPLE

The routine employed in this example is a simple loop which totals the contents of a block of data. In order to imbue it with a practical aspect, assume that program "TOTAL" computes personal expenses for a 31-day month. Data (each day's expenses) is read in from the Punched Tape Reader. The sum is printed out on the Teleprinter.

The program is written and assembled as below. To check it out a data tape, consisting of a series of 10's is prepared:

```
10 ( CR LF )  
10 ( CR LF )  
10 ( CR LF )  
:  
:
```

PAGE 0002 #01

0001	00000	NAM TOTAL	
0002	00000 000000	START NOP	
0003	00001 062162R	LDA =D-31	
0004	00002 072156R	STA CTR	
0005	00003 062163R	LDA =B5	
0006	00004 006404	CLB INB	
0007	00005 016004X	JSB .DIO.	
0008	00006 000000	ABS 0	
0009	00007 000014R	DEF *+5	
0010	00010 016006X	JSB .IOR.	
0011	00011 016001X	DST INPUT,1	
	00012 100055R		
0012	00013 016005X	JSB .RAR.	
0013	00014 066055R	LDB INPUT	INPUT THE DATA
0014	00015 046164R	ADB =B2	
0015	00016 076055R	STB INPUT	
0016	00017 036156R	ISZ CTR	
0017	00020 026003R	JMP START+3	

0019	00021	062162R	LDA =D-31	
0020	00022	072156R	STA CTR	INITIALIZE
0021	00023	016002X	DLD =F0.0	
	00024	000165R		
0022	00025	016001X	DST ANSW	
	00026	000154R		
0024	00027	016002X	DLD .MON,1	
	00030	100054R		
0025	00031	016003X	FAD ANSW	
	00032	000154R		
0026	00033	016001X	DST ANSW	
	00034	000154R		
0027	00035	066054R	SUM LDB .MON	
0028	00036	046164R	ADB =B2	ADDITION LOOP
0029	00037	076054R	STB .MON	
0030	00040	036156R	ISZ CTR	
0031	00041	026035R	JMP SUM	
0033	00042	062164R	LDA =B2	
0034	00043	006400	CLB	
0035	00044	016004X	JSB .DIO.	
0036	00045	000157R	DEF OUTPT	
0037	00046	000053R	DEF **5	
0038	00047	016002X	DLD ANSW	OUTPUT THE RESULT
	00050	000154R		
0039	00051	016006X	JSB .IOR.	
0040	00052	016007X	JSB .DTA.	
0041	00053	102077	HLT 77B	
0043	00054	000056R	.MON DEF MONTH	
0044	00055	000056R	INPUT DEF MONTH	
0045	00056	000000	MONTH BSS 62	
0046	00154	000000	ANSW BSS 2	
0047	00156	000000	CTR BSS 1	
0048			EXT .DIO...RAR...IOR...DTA.	
0049	00157	024106	OUTPT ASC 3,(F B.2)	
	00160	034056		
	00161	031051		
	00162	177741		
	00163	000005		
	00164	000002		
	00165	000000		
	00166	000000		
0050			END START	
**	NO	ERRORS*		

The "TOTAL" object tape is loaded by the Basic Control System. The debugging system is loaded next and then the library tape. The program is executed using the Debugging System by the following instructions:

10.00
H P=00053 I=102077 A=177777 B=006115 E=0 O=1

The procedure below illustrates one method for detecting errors in the program.

Dump a portion of the storage area MONTH

```
00056                                050503 000333
00060 001253 000000 000000 004267 017700 000000 053070 011770
00070 002256
```

```
B, I, 21
R, 1
I P= 00021 I=062162 A=000000 B=002154 E=0 Q=0 MA=00162 MC=177741
```

Check to see that the data has been stored in memory:

```
D,B,56,70
DUMP--BASE = 02000
00056                                050000 000010
00060 050000 000010 050000 000010 050000 000010 050000 000010
00070 050000
```

Knowing that the data has been stored in MONTH, perform the first addition:

B, I, 35

R, 21

I P= 00035 I=066054 A=050000 B=000010 E=0 O=0 MA=00054 MC=002056

Check to see that the first day's expenses have been stored at
ANSW:

D, B, 154, 155

DUMP--BASE = 02000

00154

050000 000010

The first addition was executed. Perform the remaining additions by looping:

B, I, 42

R, 35

I P= 00042 I=062164 A=050000 B=002154 E=0 O=0 MA=00164 MC=000002

Check final total in ANSW.

D, B, 154, 155

DUMP--BASE = 02000

00154

050000 000010

Here, if not previously, the error should be detected; the program does not perform more than the first addition. The label sum has been placed in the wrong instruction. It should be in location 27 preceding the "DLD . MON, I" instruction.

0001	00000		NAM TOTAL	
0002	00000	000000	START NOP	
0003	00001	062162R	LDA =D-31	
0004	00002	072156R	STA CTR	
0005	00003	062163R	LDA =B5	
0006	00004	006404	CLB, INB	
0007	00005	016004X	JSB .DIO.	
0008	00006	000000	ABS 0	
0009	00007	000014R	DEF **5	
0010	00010	016006X	JSB .IOR.	
0011	00011	016001X	DST INPUT, I	
	00012	100055R		
0012	00013	016005X	JSB .RAR.	
0013	00014	066055R	LDB INPUT	INPUT THE DATA
0014	00015	046164R	ADB =B2	
0015	00016	076055R	STB INPUT	
0016	00017	036156R	ISZ CTR	
0017	00020	026003R	JMP START+3	
0019	00021	062162R	LDA =D-31	
0020	00022	072156R	STA CTR	INITIALIZE
0021	00023	016002X	DLD =F0.0	
	00024	000165R		
0022	00025	016001X	DST ANSW	
	00026	000154R		
0024	00027	016002X	SUM DLD .MON, I	
	00030	100054R		
0025	00031	016003X	FAD ANSW	
	00032	000154R		
0026	00033	016001X	DST ANSW	
	00034	000154R		
0027	00035	066054R	LDB .MON	
0028	00036	046164R	ADB =B2	ADDITION LOOP
0029	00037	076054R	STB .MON	
0030	00040	036156R	ISZ CTR	
0031	00041	026035R	JMP SUM	
0033	00042	062164R	LDA =B2	
0034	00043	006400	CLB	
0035	00044	016004X	JSB .DIO.	
0036	00045	000157R	DEF OUTPT	
0037	00046	000053R	DEF **5	
0038	00047	016002X	DLD ANSW	OUTPUT THE RESULT
	00050	000154R		
0039	00051	016006X	JSB .IOR.	
0040	00052	016007X	JSB .DTA.	
0041	00053	102077	HLT 77B	

BINARY CODED DECIMAL FORMAT

Kennedy 1406/1506 ASCII-BCD Conversion

Symbol	BCD (octal code)	ASCII Equivalent (octal code)	Symbol	BCD (octal code)	ASCII Equivalent (octal code)
(Space)	20	040	A	61	101
!	52	041	B	62	102
#	13	043	C	63	103
\$	53	044	D	64	104
%	34	045	E	65	105
&	60	046	F	66	106
'	14	047	G	67	107
(34	050	H	70	110
)	74	051	I	71	111
*	54	052	J	41	112
+	60	053	K	42	113
,	33	054	L	43	114
-	40	055	M	44	115
.	73	056	N	45	116
/	21	057	O	46	117
0	12	060	P	47	120
1	01	061	Q	50	121
2	02	062	R	51	122
3	03	063	S	22	123
4	04	064	T	23	124
5	05	065	U	24	125
6	06	066	V	25	126
7	07	067	W	26	127
8	10	070	X	27	130
9	11	071	Y	30	131
:	15	072	Z	31	132
;	56	073	[75	133
<	76	074	\	36	134
=	13	075]	55	135
>	16	076			
?	72	077			
@	14	100			

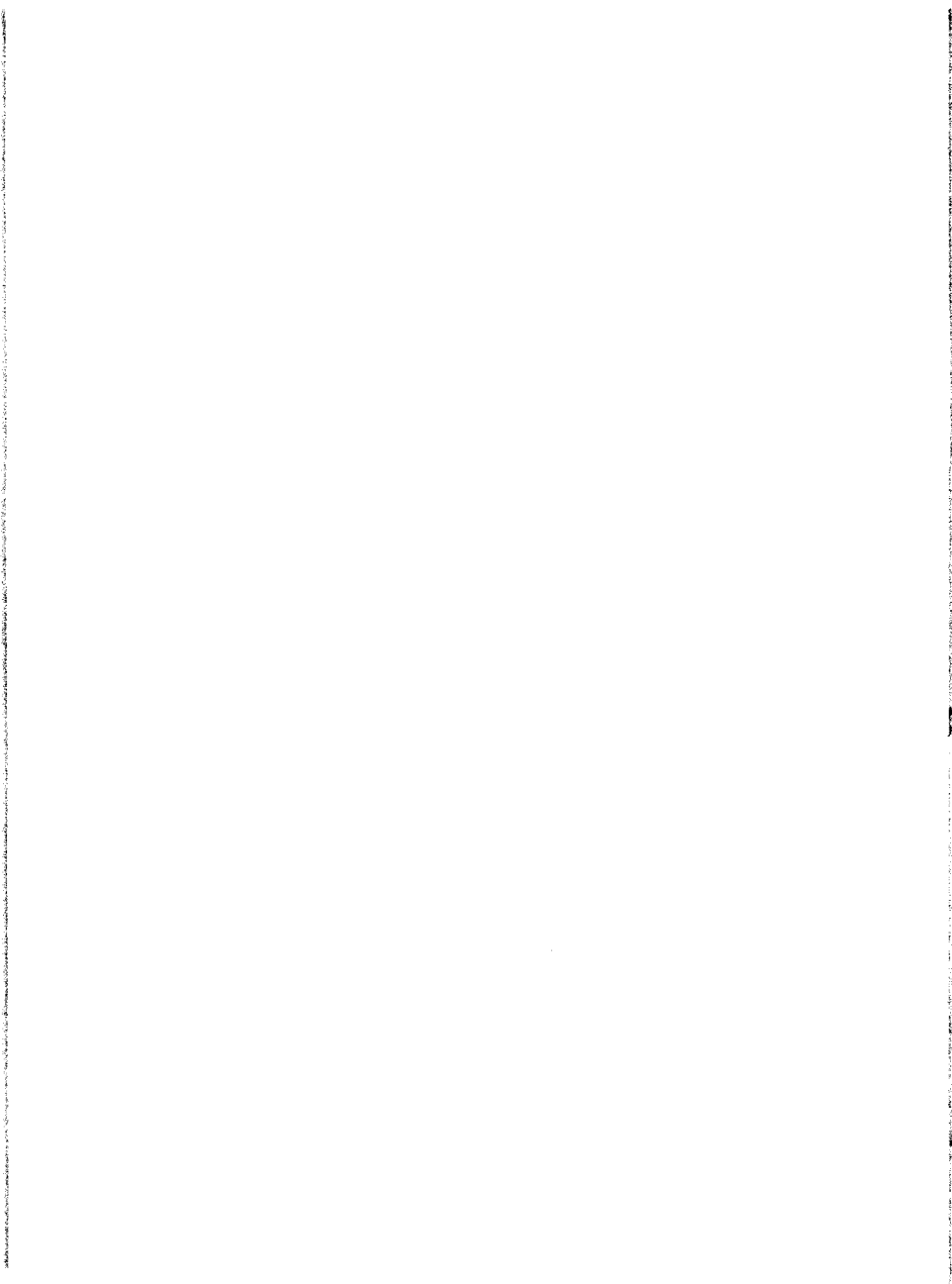
Other symbols which may be represented in ASCII are converted to spaces in BCD (20)

HP 2020A/B ASCII - BCD Conversion

Symbol	ASCII (Octal code)	BCD (Octal code)	Symbol	ASCII (Octal code)	BCD (Octal code)
(Space)	40	20	A	101	61
!	41	52	B	102	62
"	42	37	C	103	63
#	43	13	D	104	64
\$	44	53	E	105	65
%	45	34	F	106	66
&	46	60 †	G	107	67
'	47	36	H	110	70
(50	75	I	111	71
)	51	55	J	112	41
*	52	54	K	113	42
+	53	60	L	114	43
,	54	33	M	115	44
-	55	40	N	116	45
.	56	73	O	117	46
/	57	21	P	120	47
0	60	12	Q	121	50
1	61	01	R	122	51
2	62	02	S	123	22
3	63	03	T	124	23
4	64	04	U	125	24
5	65	05	V	126	25
6	66	06	W	127	26
7	67	07	X	130	27
8	70	10	Y	131	30
9	71	11	Z	132	31
:	72	15	[133	75 ‡
;	73	56]	135	55 ‡
<	74	76	,	136	77
=	75	35	-	137	32
>	76	16			
?	77	72			
@	100	14			

† BCD code of 60 always converted to ASCII code 53 (+).

‡ BCD code of 75 always converted to ASCII code 50 (() and
BCD code of 55 always converted to ASCII code 51 ()).



The Equipment Table (EQT) provides information for the input/output control routine, .IOC., and the equipment driver subroutines. The table contains an entry for each peripheral device attached to a Computer configuration.

The table is constructed as a block of entries assembled by the Prepare Control System routine. The first word of the table, at the symbolic entry point .EQT., contains the number of entries in the table. An entry in the table is referenced according to its position. The numbers 1 through 6 are reserved for Standard units (see Standard Equipment Table). The number 7_g appearing in a program refers to the 1st table entry; the number 10_g, the second, and so forth. The numbers may be in the range 7_g-74_g with the largest value being determined by the number of units of equipment available at the installation.

The 4-word entry for each device contains the following information:

The channel number of the device (10_g-76_g)

A Direct Memory Access channel indicator if pertinent

Absolute address of equipment driver subroutine

Equipment type identification code.

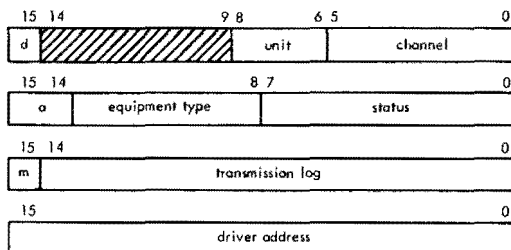
The above information is static for each installation; it is not altered by .IOC. The entry also contains dynamic information which is supplied by the equipment driver subroutine. This information includes:

Status of operation (i.e., in progress or complete)

Status of equipment

Number of characters or words transmitted when the operation is completed.

The format of the entry is as follows:



d Direct Memory Access channel indicator

1 DMA channel is to be used for each data transmission operation

0 DMA channel not required

unit Physical unit number (0-7) used to address the device if it is attached to a multi-unit controller.

channel The channel number (select code) for the physical device (also the low core location containing a JSB to the related interrupt subroutine.)

a Availability of device:

0 The device is available; the previous operation is complete.

1 The device is available; the previous operation is complete but a transmission error has been detected.

2 The device is not available for another request; the operation is in progress.

equipment type This field contains a 6-bit code that identifies the device:

00-07 – Paper Tape devices

00 2752A Teleprinter

01 2737A Punched Tape Reader

02 2753A Tape Punch

- 10-17 – Unit Record devices
- 20-37 – Magnetic Tape and Mass Storage devices
 - 20 Kennedy 1406 Incremental Tape Transport
 - 21 2020A Magnetic Tape Unit
 - 22 3030 Magnetic Tape Unit
- 40-77 – Instrumentation devices
 - 40 Data Source Interface
 - 41 Integrating Digital Voltmeter
 - 42 Guarded Crossbar Scanner
 - 43 Time Base Generator

status	The status field indicates the actual status of the device when the data transmission is complete. The contents depend on the type of device (see Status Table).
m	<p>This bit defines the mode of the data transmission:</p> <ul style="list-style-type: none"> 0 ASCII or BCD 1 Binary
transmission log	This field is a log of the number of characters or words transmitted. The value is given as a positive integer and indicates characters or words as specified in the calling sequence. The value is stored in this field only when the input/output request has been completed, therefore, when all data is transmitted or when a transmission error is detected.
driver address	Absolute address of the entry point for the associated driver subroutine for the device.

STATUS TABLE

Device \ Status Bit	7	6	5	4	3	2	1	0
2752A Teleprinter			End of Input Tape					
2737A Punched Tape Reader			End of Tape					
2753A Tape Punch			Tape Supply Low					
Kennedy 1406 Incremental Tape Transport			End of Tape		BT			DB
2020A Magnetic Tape Unit	EOF	ST	End of Tape	TE	I/O R	NW	PA	DB
3030 Magnetic Tape Unit	EOF	ST	End of Tape	TE	I/O R	NW	PA	DB

BT = Broken Tape

DB = Device Busy

EOF = End of File

ST = Start of Tape

TE = Timing Error

I/O R = I/O Reject

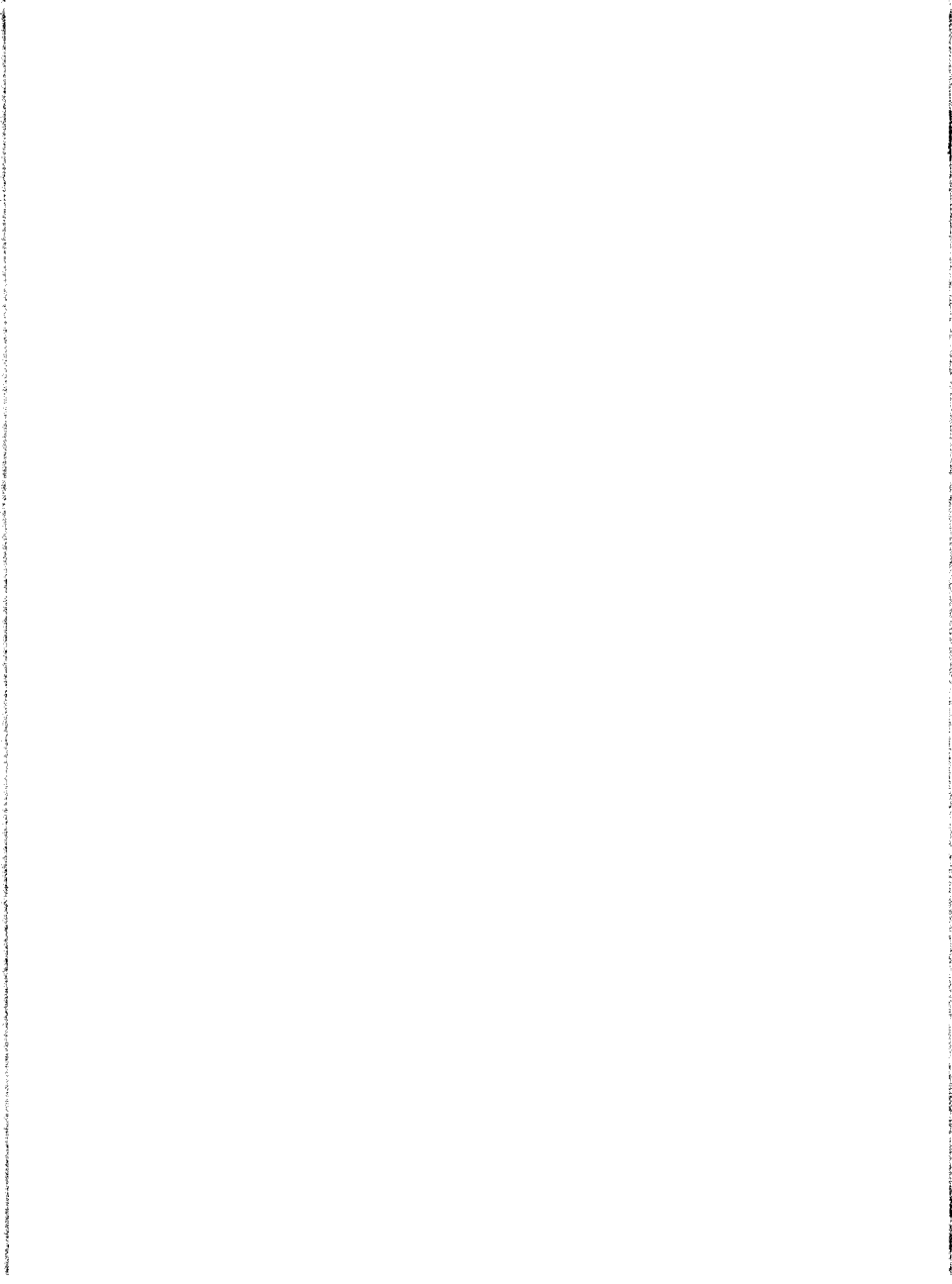
NW = No Write (write enable ring missing or tape unit is rewinding)

PA = Parity Error

The Standard Unit Equipment Table (SQT) allows reference to input/output devices designated as Standard units. The Table contains six 1-word entries. Each entry corresponds to a particular Standard unit and contains a pointer to the Equipment Table. The Standard units are as follows:

<u>Number</u>	<u>Name</u>
1	Keyboard Input
2	Teleprinter Output
3	Program Library
4	Punch Output
5	Input
6	List Output

The number defines the position in the SQT at which the device is listed. Each Standard unit may be a different device, or a single physical device may represent several Standard units. The value of the pointer in the SQT is the position of the physical unit's entry in the EQT, with the lowest value being 7₈ (see EQT).



IOC with Output Buffering is an extension of the standard version and provides for automatic stacking and buffering of all output and function requests. This involves moving an output request and associated buffer into available memory and adding the request location into a thread of stacked requests for the referenced unit. At the completion of an output operation, the next entry in the stack for the unit is initiated by IOC. The processing of output/function requests for a particular unit is according to the order of the requests (first in/first out). This version of IOC requires the use of the program MEMORY to perform the allocation and release of blocks of available memory. If available memory is exhausted when an allocation is attempted, IOC repeats the call until space is made available, i. e., previous blocks are released.

PRIORITY OUTPUT

A "Priority" write or function request has been added for use with the Buffered version of IOC. A "Priority" request is processed immediately without the request and buffer being moved to available memory. The current operation in the stack for the referenced unit is suspended, the "Priority" request processed, and the suspended operation re-initiated. The "Priority" feature is useful for writing messages or diagnostics for immediate action or for performing output without reserving a segment of available memory for request/buffer storage. (All output performed by the BCS Relocating Loader is done as Priority requests because of the latter reason.) (NOTE: If two (2) or more Priority requests are called in immediate succession (without intervening status checks), the last requested operation is performed with the previous ones being "lost".)

A "Priority" request (i.e., Write function) is indicated by setting bit 09 of Word 2 of the request call = 1. Bit 09 = 0 means normal operation with the Standard IOC and means the request will be stacked and buffered with the extended version.

Example: "Priority" Write to Teleprinter

```
JSB   .IOC.  
OCT   21002  
JMP   REJ  
DEF   BUFFR  
DEC   -37
```

OPERATING ENVIRONMENT

IOC with Output Buffering provides for writing a data block on more than one output device in parallel and does not restrict output rates to the lowest speed device. Because all requests and buffers are moved into available memory for subsequent processing, peak load output processing is not delayed due to device speed or saturated buffer storage within the bounds of user programs. System I/O saturation occurs when available memory is exhausted.

RESTRICTIONS

The routines used to allocate/release blocks in available memory and to initiate stacked output requests operate with the Interrupt System disabled. Therefore, the use of medium/high speed synchronous I/O devices under program control is not recommended because of possible data loss (e.g., HP 2020 A/B Magnetic Tape).

An I/O driver routine operating under the extended version of IOC may not be used to control more than one like device. This is because the buffering control routine in IOC only checks for stacked requests referencing the unit on which an operation was just completed.

HALT CONDITIONS

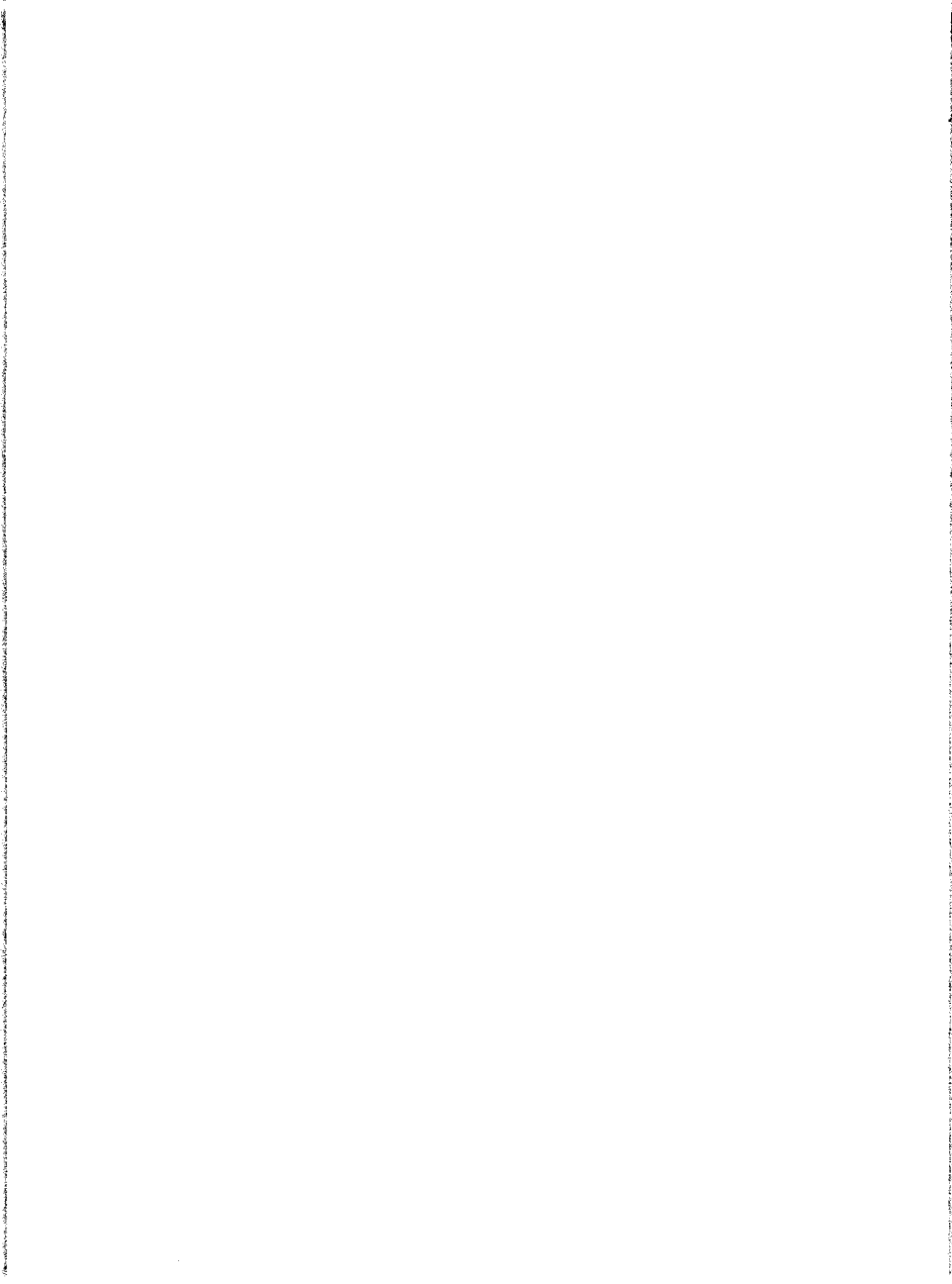
Irrecoverable error conditions are identical to the Standard version of IOC. The location of the halt is at the entry point "IOERR". These conditions are:

<u>A-Register</u>	<u>B-Register</u>	<u>Meaning</u>
Ø	Location at Request	Request Code Illegal
1	Location at Request	Unit Reference Illegal
Ø	Ø	Write request for an input only device.

I/O ERROR CONDITIONS

The routine .BUFR in the version of IOC with Output Buffering checks for error conditions of the end of each output operation. If any error conditions and End-of-Tape or Tape Supply Low, etc. conditions are present, IOC halts to allow the condition to be corrected. Processing is continued by pressing RUN.

- Halt: (T) = 1Ø2Ø7Ø
 (A) = Word 2 of EQT entry (Status word)
 (B) = Hardware I/O address of unit



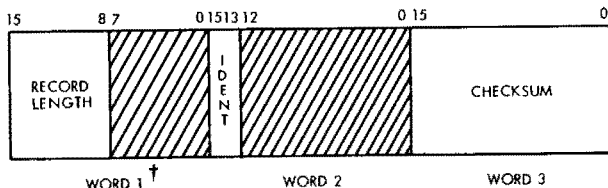
RELOCATABLE TAPE FORMAT

E

NAM RECORD

CONTENT

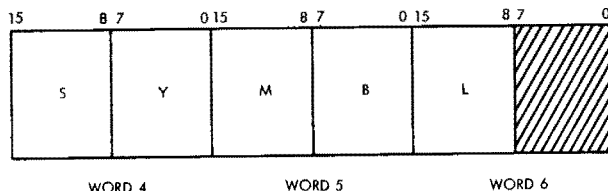
EXPLANATION



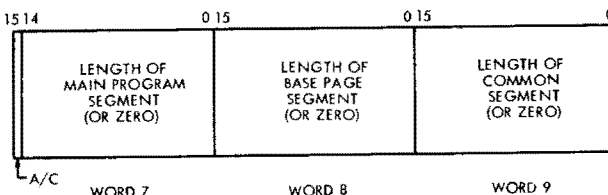
RECORD LENGTH = 9 WORDS

IDENT = 001

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS IN RECORD EXCLUDING WORDS 1 AND 3.

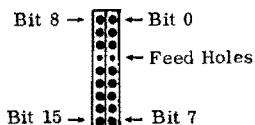


SYMBOL: FIVE CHARACTER NAME OF PROGRAM



A/C: BINARY TAPE PROCESSOR
= 0 IF ASSEMBLER PRODUCED
= 1 IF COMPILER PRODUCED

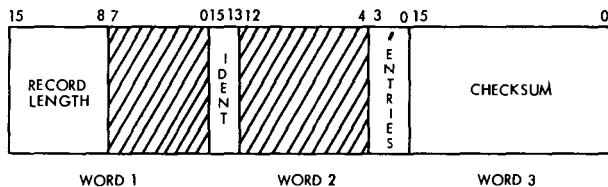
† Each word represents two frames arranged as follows:



ENT RECORD

CONTENT

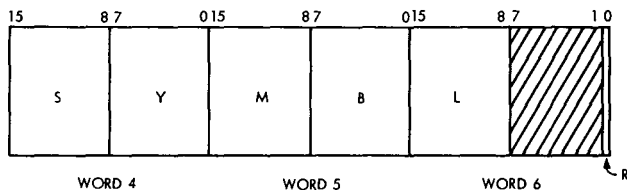
EXPLANATION



RECORD LENGTH = 7-59 WORDS

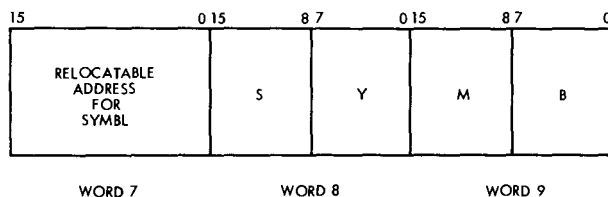
IDENT = 010

ENTRIES: 1 to 14 ENTRIES PER PROGRAM; EACH ENTRY IS FOUR WORDS LONG.

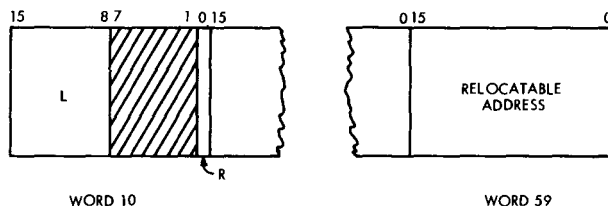


SYMBL: 5 CHARACTER ENTRY POINT SYMBOL

R: RELOCATION INDICATOR
= 0 IF PROGRAM RELOCATABLE
= 1 IF BASE PAGE RELOCATABLE



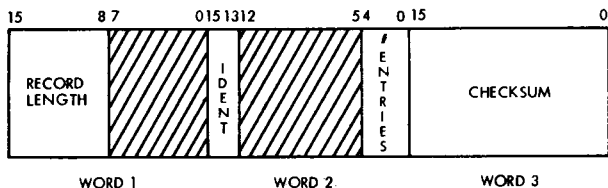
WORDS 4 THROUGH 7 ARE REPEATED FOR EACH ENTRY POINT SYMBOL.



EXT RECORD

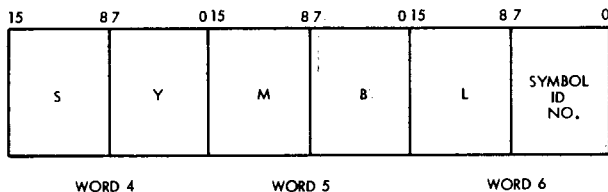
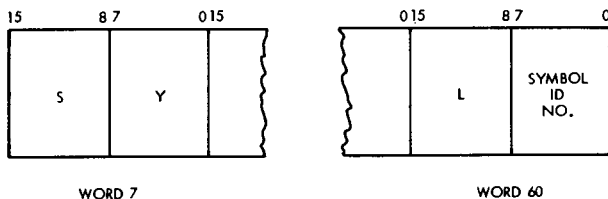
CONTENT

EXPLANATION

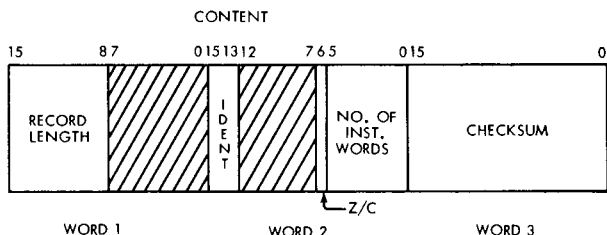


RECORD LENGTH = 6-60 WORDS

IDENT = 100

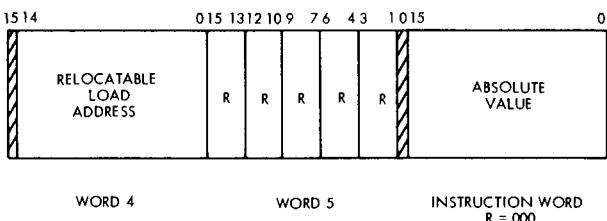
ENTRIES: 1 TO 19 PER
RECORD; EACH ENTRY
IS THREE WORDS LONGSYMBL: 5 CHARACTER
EXTERNAL SYMBOLSYMBOL ID, NO.: NUMBER
ASSIGNED TO SYMBL FOR
USE IN LOCATING
REFERENCE IN BODY
OF PROGRAM.WORDS 4 THROUGH 6 REPEATED
FOR EACH EXTERNAL
SYMBOL (MAXIMUM OF
19 PER RECORD).

DBL RECORD



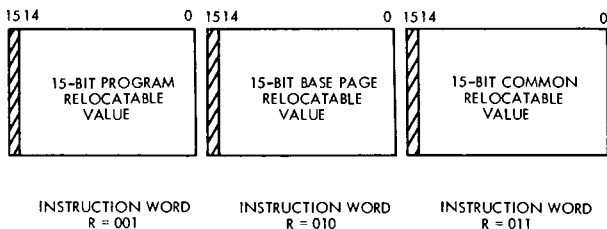
EXPLANATION

RECORD LENGTH = 5-60 WORDS
 IDENT = 011
 Z/C: BASE/CURRENT PAGE LOADING
 = 0 FOR BASE PAGE
 = 1 FOR CURRENT PAGE
 NO. OF INST. WORDS: 1 TO 45
 LOADABLE INSTRUCTION
 WORDS PER RECORD



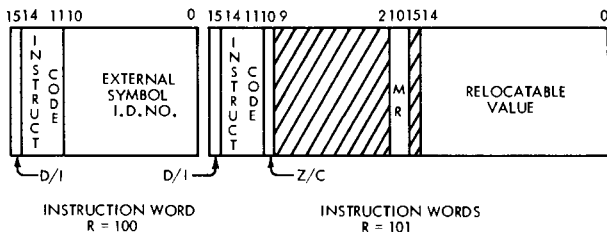
RELOCATABLE LOAD ADDRESS:
 STARTING ADDRESS FOR
 LOADING THE INSTRUCTIONS
 WHICH FOLLOW.

R's: RELOCATION INDICATORS:
 000 = ABSOLUTE
 001 = 15-BIT PROGRAM
 RELOCATABLE
 010 = 15-BIT BASE PAGE
 RELOCATABLE
 011 = 15-BIT COMMON
 RELOCATABLE
 100 = EXTERNAL REFERENCE
 101 = MEMORY REFERENCE



R₁ IS RELOCATION INDICATOR FOR
 INSTRUCTION WORD₁; R₂, FOR
 INSTRUCTION WORD₂; ETC.-MEMORY
 REFERENCE INSTRUCTIONS USE
 TWO WORDS, WITHIN THE TWO-
 WORD GROUP, "MR" INDICATES
 RELOCATABILITY OF OPERAND
 SPECIFIED IN SECOND WORD:

00 = PROGRAM RELOCATABLE
 01 = BASE PAGE RELOCATABLE
 10 = COMMON RELOCATABLE



D/I: INDIRECT ADDRESSING

0 = DIRECT
 1 = INDIRECT

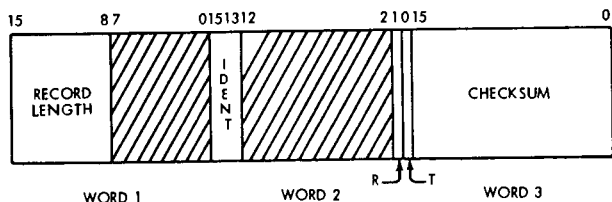
Z/C: BASE/CURRENT PAGE LOCA-
 TION OF OPERAND ADDRESS
 AS DETERMINED BY LOADER.

0 = BASE PAGE
 1 = CURRENT PAGE

END RECORD

CONTENT

EXPLANATION



RECORD LENGTH = 4 WORDS

IDENT = 101

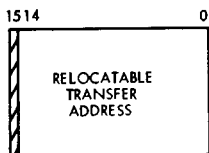
R: RELOCATION INDICATOR
FOR TRANSFER ADDRESS

= 0 IF PROGRAM RELOCATABLE
= 1 IF BASE PAGE
RELOCATABLE

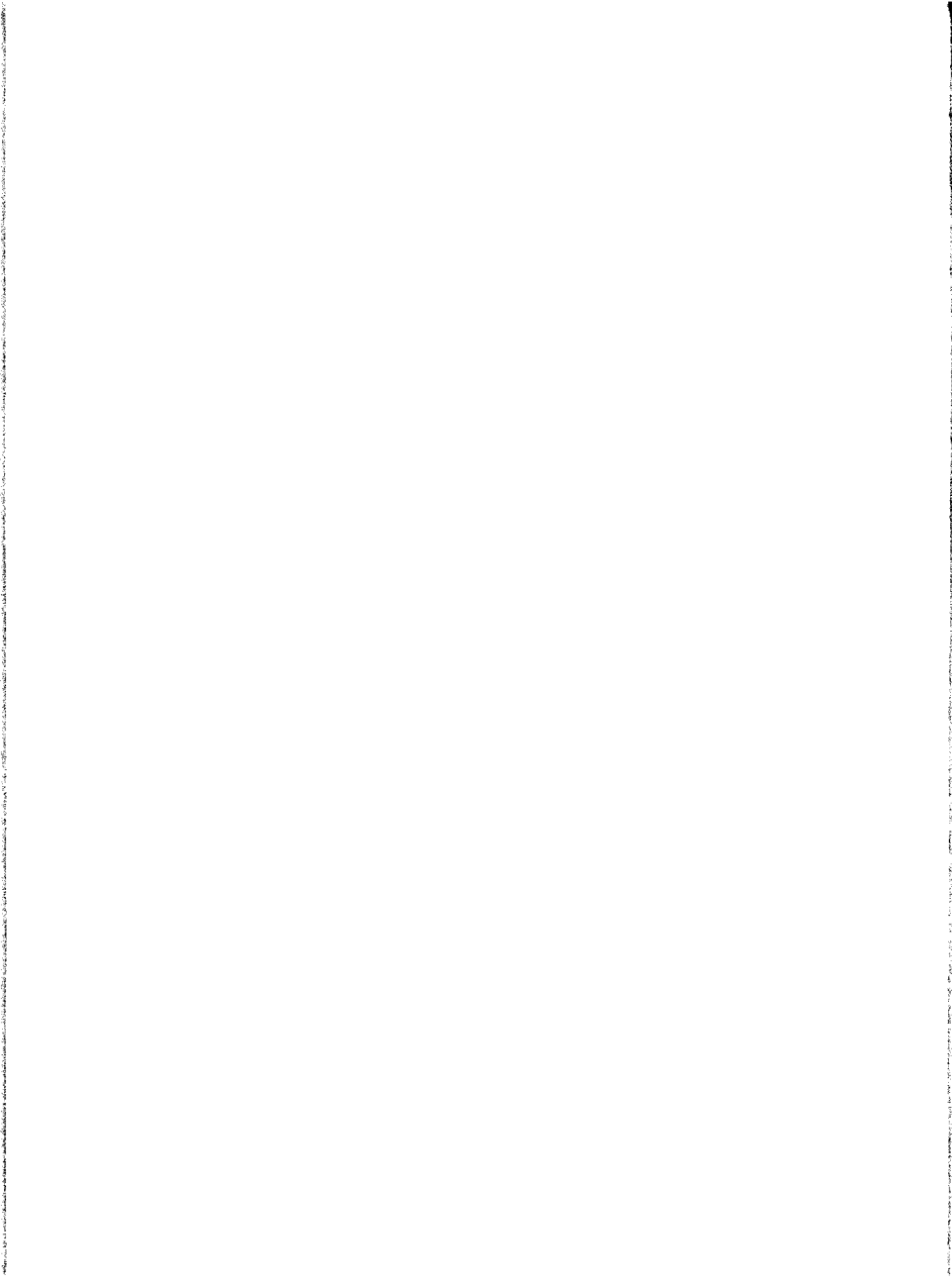
T: TRANSFER ADDRESS
INDICATOR

= 0 IF NO TRANSFER
ADDRESS IN RECORD

= 1 IF TRANSFER ADDRESS
PRESENT

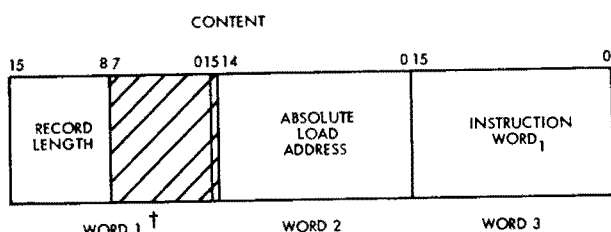


WORD 4



ABSOLUTE TAPE FORMAT

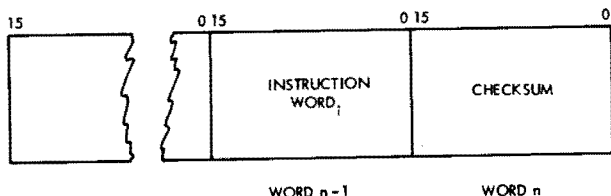
F



EXPLANATION

RECORD LENGTH = NUMBER OF WORDS IN RECORD EXCLUDING WORDS 1 AND 2 AND THE LAST WORD.

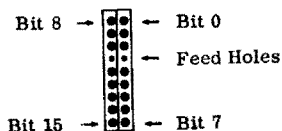
ABSOLUTE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW

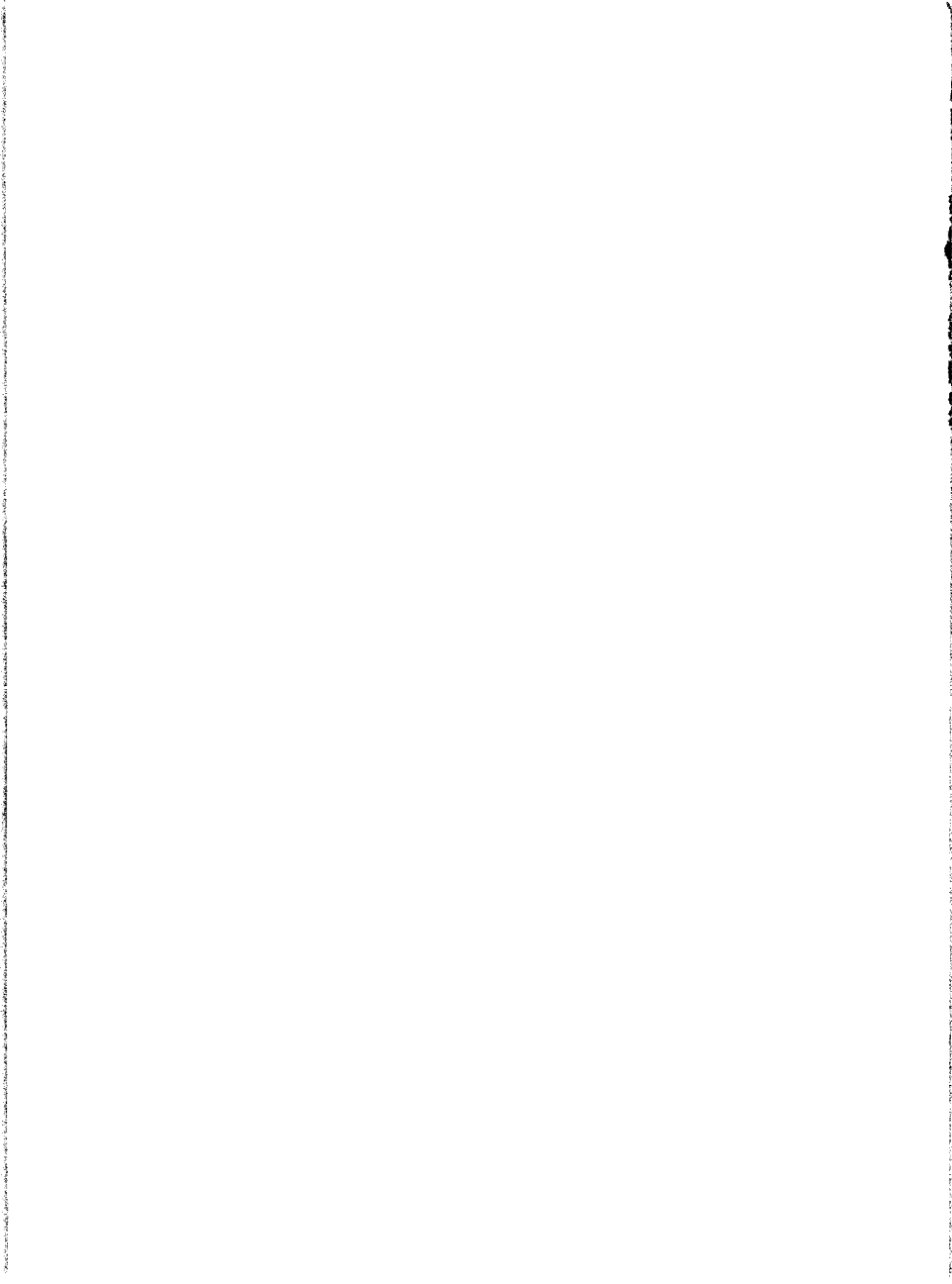


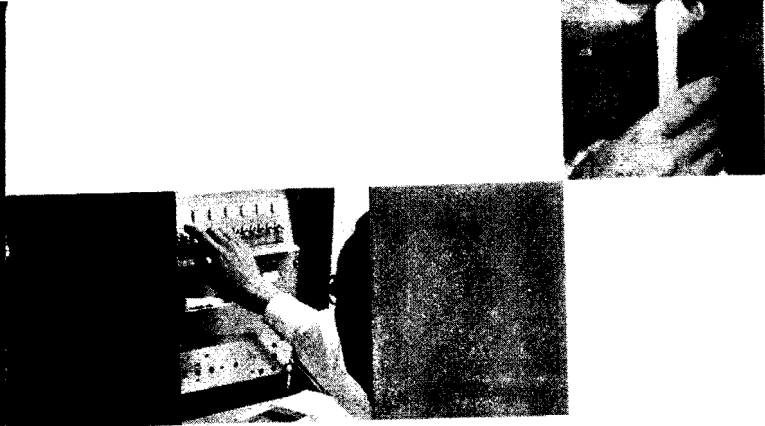
INSTRUCTION WORDS: ABSOLUTE INSTRUCTIONS OR DATA

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS EXCEPT FIRST AND LAST

†Each word represents two frames arranged as follows:







FORTRAN Reference Manual

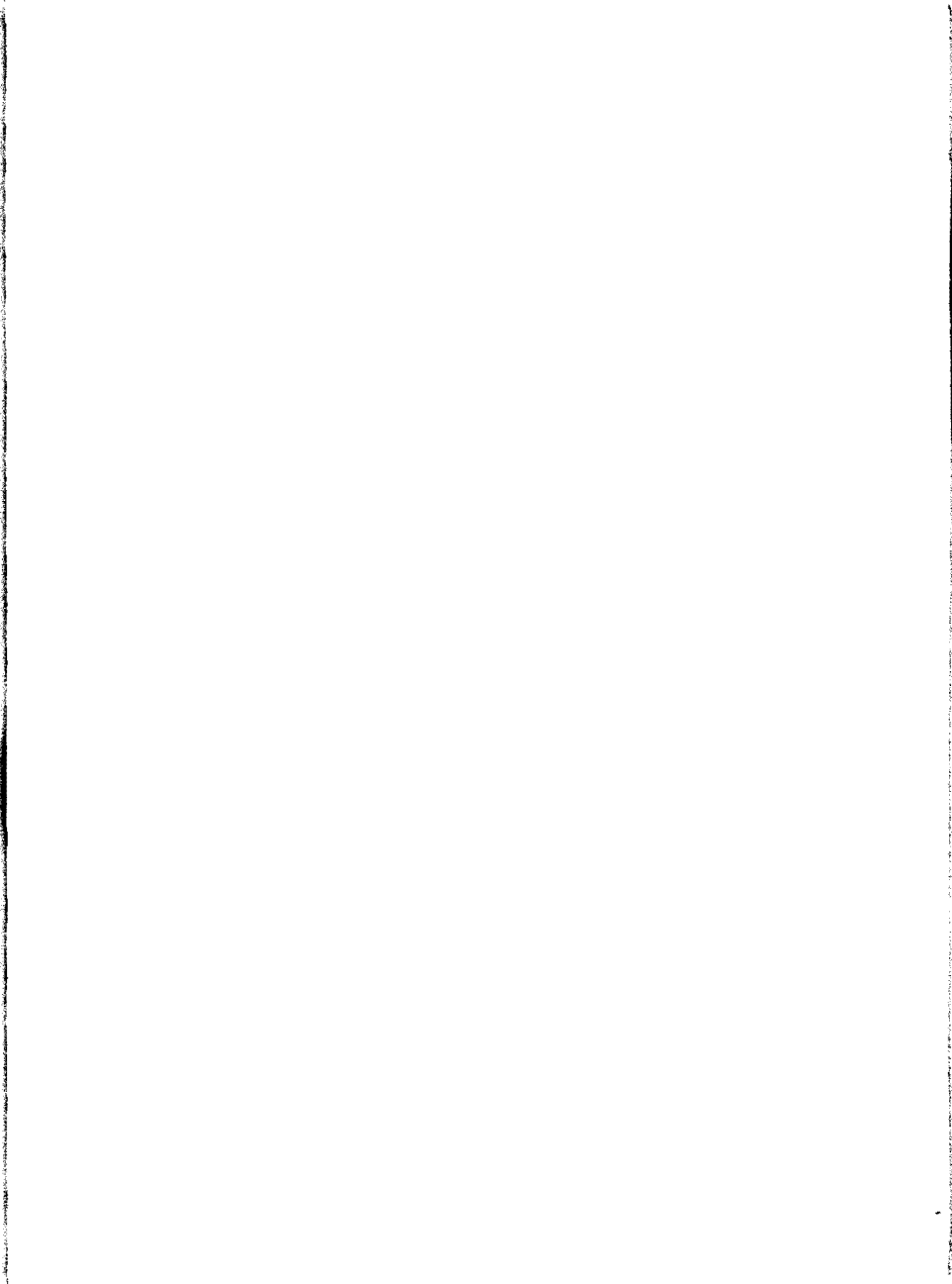
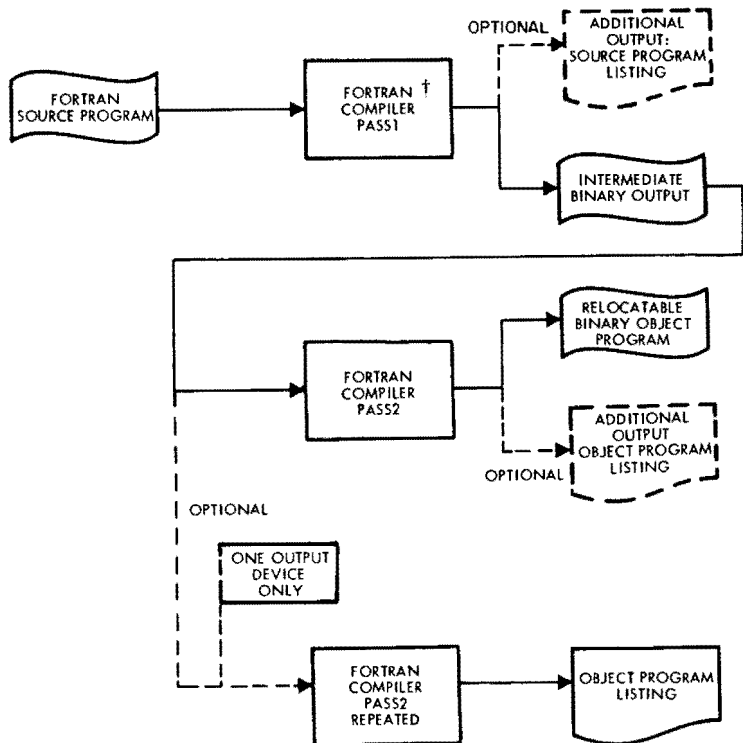


TABLE OF CONTENTS

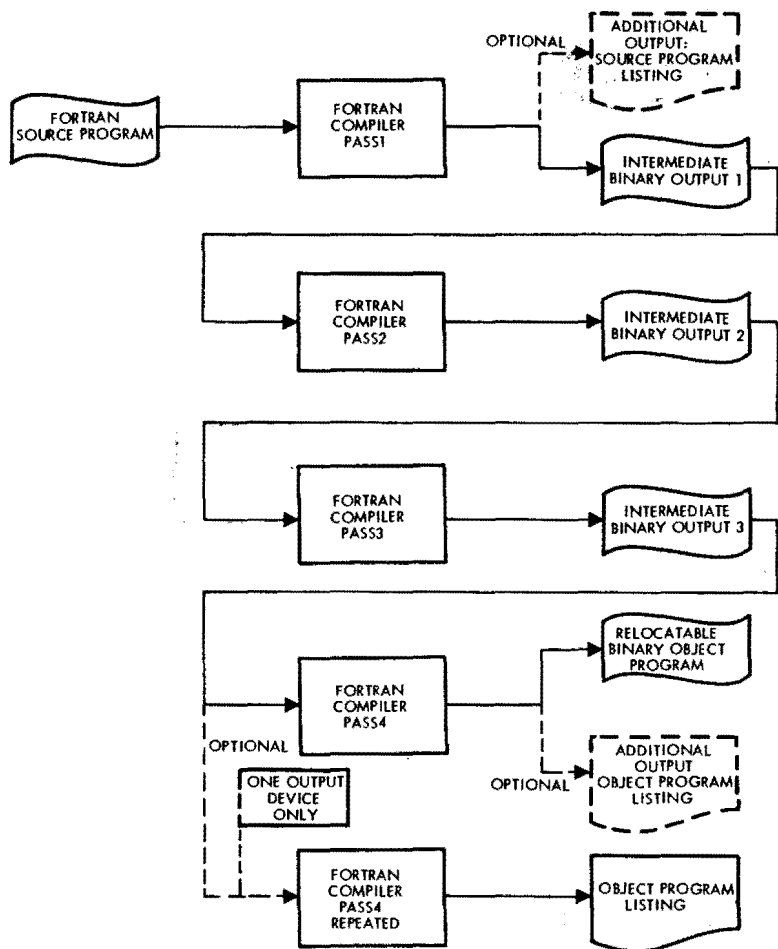
INTRODUCTION		v
CHAPTER 1	PROGRAM FORM	1-1
	1.1 Character Set	1-1
	1.2 Lines	1-2
	1.3 Coding Form	1-3
CHAPTER 2	ELEMENTS OF HP FORTRAN	2-1
	2.1 Data Type Properties	2-1
	2.2 Constants	2-2
	2.3 Variables	2-3
	2.4 Arrays	2-5
	2.5 Expressions	2-6
	2.6 Statements	2-7
CHAPTER 3	ARITHMETIC EXPRESSIONS AND ASSIGNMENT STATEMENTS	3-1
	3.1 Arithmetic Expressions	3-1
	3.2 Assignment Statements	3-4
	3.3 Masking Operations	3-5
CHAPTER 4	SPECIFICATIONS STATEMENTS	4-1
	4.1 Dimension	4-1
	4.2 Common	4-2
	4.3 Equivalence	4-5
CHAPTER 5	CONTROL STATEMENTS	5-1
	5.1 GO TO Statements	5-1
	5.2 IF Statements	5-2
	5.3 DO Statements	5-3
	5.4 CONTINUE	5-9
	5.5 PAUSE	5-9
	5.6 STOP	5-9
	5.7 END	5-10
	5.8 END\$	5-10

CHAPTER 6	MAIN PROGRAM, FUNCTIONS, AND SUBROUTINES	6-1
6.1	Argument Characteristics	6-1
6.2	Main Program	6-2
6.3	Subroutine Program	6-2
6.4	Subroutine Call	6-4
6.5	Function Subprogram	6-5
6.6	Function Reference	6-7
6.7	Statement Function	6-9
6.8	Basic External Functions	6-11
6.9	RETURN and END	6-12
CHAPTER 7	INPUT/OUTPUT LISTS AND FORMAT CONTROL	7-1
7.1	Input/Output Lists	7-1
7.2	FORMAT Statement	7-4
7.3	FORMAT Statement Conversion Specifications	7-4
7.4	Free Field Input	7-16
CHAPTER 8	INPUT/OUTPUT STATEMENTS	8-1
8.1	Unit-Reference	8-1
8.2	Formatted READ, WRITE	8-2
8.3	Unformatted READ, WRITE	8-3
8.4	Auxiliary Input/Output Statements	8-3
CHAPTER 9	COMPILER INPUT/OUTPUT	9-1
9.1	Control Statement	9-1
9.2	Source Program	9-2
9.3	Binary Output	9-2
9.4	List Output	9-2
9.5	Operating Instructions: Paper Tape	9-4
9.6	Operating Instructions: Magnetic Tape	9-10
9.7	Diagnostic Messages	9-12
9.8	Object Program Loading	9-13
9.9	Object Program Diagnostic Messages	9-17
APPENDIX A	HP Character Set	A-1
APPENDIX B	FORTRAN Statements and Functions	B-1
APPENDIX C	Assembly Language Subprograms	C-1
APPENDIX D	Instrument Formats	D-1
APPENDIX E	Sample Program	E-1



†When compiling with the magnetic tape system, operator intervention ceases after Pass 1 has been loaded.

8K MEMORY FORTRAN COMPILATION PROCESS



4K MEMORY FORTRAN COMPILATION PROCESS

INTRODUCTION

The FORTRAN compiler system accepts as input, a source program written according to American Standard Basic FORTRAN specifications; it produces as output, a relocatable binary object program which can be loaded and executed under control of the HP Basic Control System.

In addition to the ASA Basic FORTRAN language, HP FORTRAN provides a number of features which expand the flexibility of the system. Included are:

- Free Field Input: Special characters included with ASCII input data direct its formatting; a FORMAT statement need not be specified in the source program.

- Specification of heading and editing information in the FORMAT statement through use of the "..." notation; permits alphanumeric data to be read or written without giving the character count.

- Array declaration within a COMMON statement.

- Redefinition of its arguments and common areas by a function subprogram.

- Interpretation of an END statement as a RETURN statement.

- Basic External Functions which perform masking (Boolean) operations.

- Two-branch IF statement.

- Octal constants.

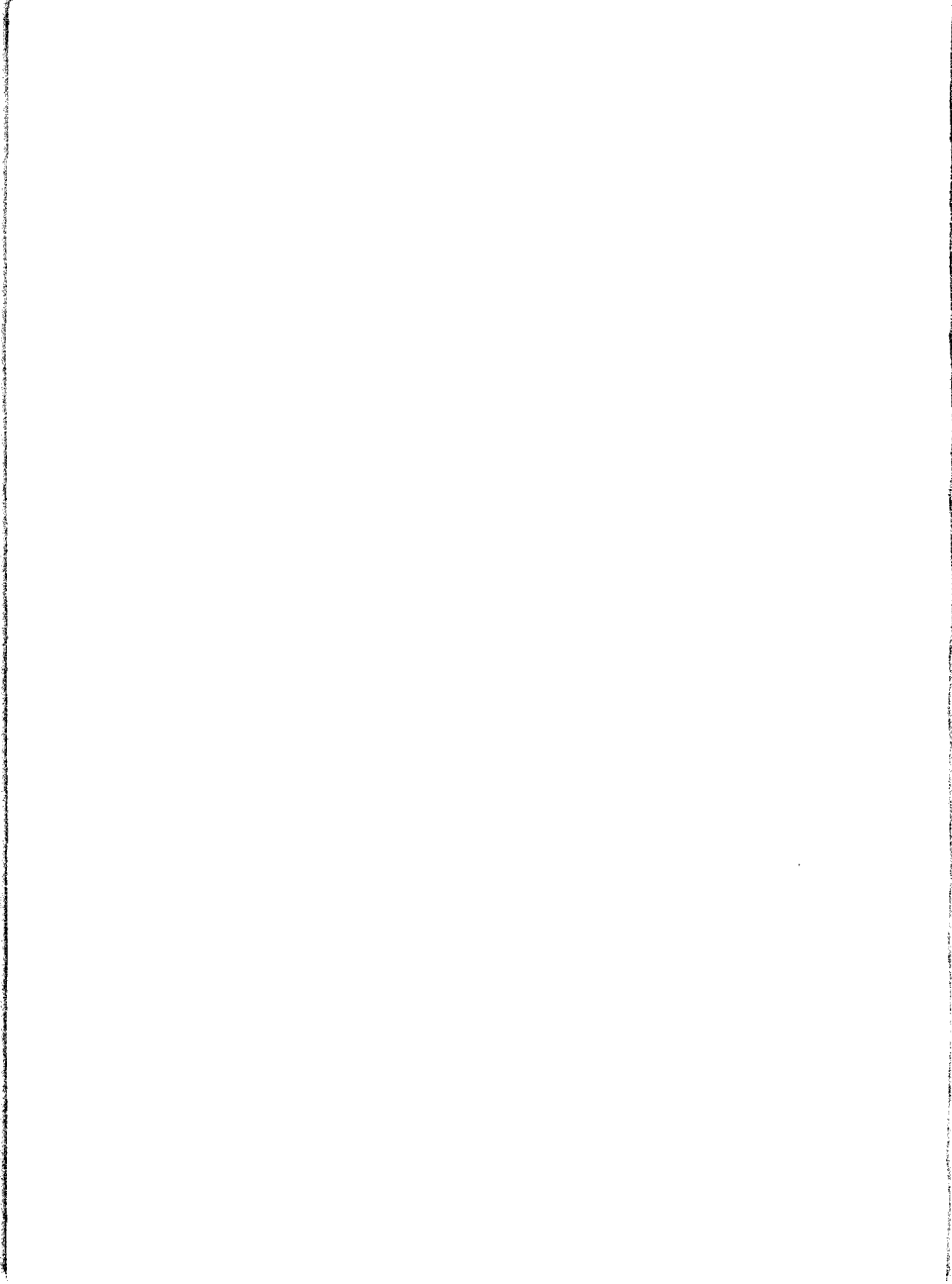
The paper tape version of the compiler operates in two or four passes depending on the size of memory. For an 8K system, the compiler produces a source listing and an intermediate binary tape in the first pass. This intermediate tape serves as input to the second pass. The second pass produces the relocatable object program tape and a listing of the program in assembly level language. If only one output device is available the last pass is repeated to produce the listing. For the 4K system, two additional passes are introduced before the pass producing the relocatable program tape. For these passes the intermediate binary output of the previous pass becomes the input for the current pass.

The magnetic tape version of the compiler uses the scratch area for storage of intermediate binary code. Pass 1 of FORTRAN writes the intermediate program. At the end of Pass 1, FORTRAN calls the Inter-Pass Loader; it searches for and loads Pass 2 of FORTRAN. Pass 2 spaces forward to the scratch area, processes the intermediate code and produces output on the punch and list devices as requested.

The minimum equipment configuration required to compile a program on the Computer is as follows:

- 2116B, 2115A, or 2114A Computer with 4K memory

- 2752A Teleprinter.



A FORTRAN program is constructed of characters grouped into lines and statements.

1.1 CHARACTER SET

The program is written using the following characters:

Alphabetic:	A through Z
Numeric:	0 through 9
Special:	
	Space
=	Equals
+	Plus
-	Minus
*	Asterisk
/	Slash
(Left Parenthesis
)	Right Parenthesis
,	Comma
.	Decimal Point
\$	Dollar Sign
"	Quotation mark

Spaces may be used anywhere in the program to improve appearance; they are significant only within heading data of FORMAT statements and, in lieu of other information, in the first six positions of a line.

In addition to the above set which is used to construct source language statements, certain characters have special significance when appearing with ASCII input data. They are the following:

space,	Data item delimiters
/	Record terminator
+ -	Sign of item
.E+-	Floating point number
@	Octal integer
"..."	Comments

Details on the input data character set are given in Chapter 7.

1.2 LINES

A line is a string of up to 72 characters. On paper tape, each line is terminated by a return, **CR**, followed by a line feed, **LF**. This terminator may be in any position following the statement information or comment contained in the line.

Statements

A statement may be written in an initial line and up to five continuation lines. The statement may occupy positions 7 through 72 of these lines. The initial line contains a zero or blank in position 6. A continuation line contains any character other than zero or space in position 6 and may not contain a C in position 1.

Statement Labels

A statement may be labeled so that it may be referred to in other statements. A label consists of one to four numeric digits placed in any of the first five positions of a line. The number is unsigned and in the range of 1 through 9999. Imbedded spaces and leading zeros are ignored. If no label is used, the first five positions of the statement line must be blank. The statement label or blank follows the **CR** **LF** terminator of the previous line.

Comments

Lines containing comments may be included with the statement lines; the comments are printed along with the source program listing. A comment line requires a C in position 1 and may occupy positions 2 through 72. If more than one line is used, each line requires a C indicator. Each comment line is terminated with a **CR** and **LF**.

Control Statement

The first statement of a program is the control statement; it defines the output to be produced by the FORTRAN compiler. The following options are available:

Relocatable binary - The program is to be loaded by the loader of the Basic Control System.

Source Listing output - A listing of the source program is produced during the first pass of compiler operation.

Object Listing output – A list of the object program is produced during the last pass of compiler operation.

The control statement must be followed by the **CR** **LF** terminator.

End Line

Each subprogram is terminated with an end line which consists of blanks in positions 1 through 6 and the letters E, N, and D located in any of the positions 7 through 72. The special end line, **END\$**, signifies the end of five or less programs being compiled at one time. The end line is terminated by **CR** **LF**.

1.3 CODING FORM

The FORTRAN coding form is shown below. Columns 73-80 may be used to indicate a sequence number for a line; they must not be punched on paper tape. All other columns of the form conform with line positions for paper tape.

A blank sheet of graph paper with a grid pattern. The vertical axis is labeled "STATEMENT" and has numbers from 0 to 60. The horizontal axis is labeled "DATE" and has numbers from 0 to 60. There are also labels for "PROGRAMMER" and "NOONAN".

4425

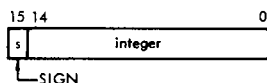
HP FORTRAN processes two types of data. They differ in mathematical significance, constant format, and symbolic representation. The two types are real and integer quantities.

2.1 DATA TYPE PROPERTIES

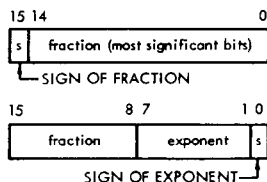
Integer and real data quantities have different ranges of values.

An integer quantity has an assumed fixed decimal point. It is represented by a 16-bit computer word with the most significant bit as the sign and the assumed decimal point on the right of the least significant bit.

An integer quantity has a range of -2^{15} to $2^{15} - 1$.



A real quantity has a floating decimal point; it consists of a fractional part and an exponent part. It is represented by two 16-bit computer words; the exponent and its sign are eight bits; the fraction and its sign are twenty-four bits.



It has a range in magnitude of approximately 10^{-38} to 10^{38} and may assume positive, negative, or zero values. If the fraction is negative, the number is in two's complement form. A zero

value is stored as all zero bits. Precision is approximately seven decimal digits.

2.2 CONSTANTS

A constant is a value that is always defined during execution and may not be redefined. Three types of constants are used in HP FORTRAN: integer, octal (treated as integer), and real. The type of constant is determined by its form and content.

Integer

An integer constant consists of a string of up to five decimal digits. If the range -32768 to 32767 (-2^{15} to $2^{15}-1$) is exceeded, a diagnostic is provided by the compiler.

Examples:

8364	5932
1720	9
1872	31254
125	1
3653	30000

Octal

Octal constants consist of up to six octal digits followed by the letter B. The form is:

$$n_1 n_2 n_3 n_4 n_5 n_6 B$$
$$n_1 \text{ is } 0 \text{ or } 1$$
$$n_2 - n_6 \text{ are } 0 \text{ through } 7$$

If the constant exceeds six digits, or if a non-octal digit appears, the constant is treated as zero and a compiler diagnostic is provided.

Examples:

7677B	7631B
3270B	5B
3520B	75026B
175B	177776B
567B	177777B

Real

Real constants may be expressed as an integer part, a decimal point, and a decimal fraction part. The constant may include an exponent, representing a power of ten, to be applied to the preceding quantity. The forms of real constants are:

$n.n \quad n. \quad .n \quad n.nE\pm e \quad n.E\pm e \quad .nE\pm e$

n is the number and e is the exponent to the base ten. The plus sign may be omitted for a positive exponent. The range of e is 0 through 38. When the exponent indicator E is followed by a $+$ or $-$ sign, then all digits between the sign and the next operator or delimiter are assumed to be part of the exponent expression, e .

If the range of the real constant is exceeded, the constant is treated as zero and a compiler diagnostic message occurs.

Examples:

4.512	4.5E2
4.	.45E+3
.512	4.5E-5
4.0	0.5
4.E-10	.5E+37
1.	10000.0

2.3 VARIABLES

A variable is a quantity that may change during execution; it is identified by a symbolic name. Simple and subscripted variables are recognized. A simple variable represents a single quantity; a subscripted variable represents a single quantity (element) within an array of quantities. Variables are identified by one to five alphanumeric characters; the first character must be alphabetic.

The type of variable is determined by the first letter of the name. The letters I, J, K, L, M, and N, indicate an integer (fixed point) variable; any other letter indicates a real (floating point) variable. Spaces imbedded in variable names are ignored.

Simple Variable

A simple variable defines the location in which values can be stored. The value specified by the name is always the current value stored in that location.

Examples:

<u>Integer</u>	<u>Real</u>
I	ALPHA
JAIME	G13
K9	DOG
MIL	XP2
NIT	GAMMA

Subscripted Variable

A subscripted variable defines an element of an array; it consists of an alphanumeric identifier with one or two associated subscripts enclosed in parentheses. The identifier names the array; the subscripts point to the particular element. If more than two subscripts appear, a compiler diagnostic message is given.

Subscripts may be integer constants, variables, or expressions; they may have the form (exp_1, exp_2) , where exp_i is one of the following:

$c*v+k$	$v-k$
$c*v-k$	v
$c*v$	k
$v+k$	

where c and k are integer constants and v is a simple integer variable.

Examples:

<u>Integer</u>	<u>Real</u>
I(J, K)	A(J)
LAD(3, 3)	BACK(M+5, 9)
MAJOR (24*K, I+5)	OP45(4*I)
NU (K+2)	RADI (IDEG)
NEXT (N*5)	VOLTI (I, J)

2.4 ARRAYS

An array is an ordered set of data of one or two dimensions; it occupies a block of successive memory locations. It is identified by a symbolic name which may be used to refer to the entire array. An array and its dimensions must be declared at the beginning of the program in a DIMENSION or COMMON statement. The type of an array is determined by the first letter of the array name. The letters I, J, K, L, M, and N, indicate an integer array; any other letter indicates a real array.

Each element of an array may be referred to by the array name and the subscript notation. Program execution errors may result if subscripts are larger than the dimensions initially declared for the array, however, no diagnostic messages are issued.

Array Structure

Elements of arrays are stored by columns in ascending order of storage locations. An array declared as SAM(3,3), would be structured as:

Columns			
Rows	SAM(1, 1)	SAM(1, 2)	SAM(1, 3)
	SAM(2, 1)	SAM(2, 2)	SAM(2, 3)
	SAM(3, 1)	SAM(3, 2)	SAM(3, 3)

and would be stored as:

m	SAM(1, 1)
m+1	SAM(2, 1)
m+2	SAM(3, 1)
m+3	SAM(1, 2)
m+4	SAM(2, 2)
m+5	SAM(3, 2)
m+6	SAM(1, 3)
m+7	SAM(2, 3)
m+8	SAM(3, 3)

The location of an array element with respect to the first element is a function of the subscripts, the first dimension, and the type of the array. Addresses are computed modulo 2^{15} .

Given $\text{DIMENSION } A(L, M)$, the memory location of $A(i, j)$ with respect to the first element, A , of the array, is given by the equation:

$$l = A + \left[i - 1 + L(j - 1) \right] * s$$

The quantity in braces is the expanded subscript expression. The element size, s , is the number of storage words required for each element of the array: for integer arrays, $s = 1$; for real arrays, $s = 2$.

Array Notation

The following subscript notations are permitted for array elements:

For a two-dimensional array, $A(d_1, d_2)$:

$A(I, J)$	implies $A(I, J)$
$A(I)$	implies $A(I, 1)$
A	implies $A(1, 1)$ †

For a single-dimension array, $A(d)$

$A(I)$	implies $A(I)$
A	implies $A(1)$

The elements of a single-dimension array, $A(d)$, however, may not be referred to as $A(I, J)$. A diagnostic message is given by the compiler if this is attempted.

2.5 EXPRESSIONS

An expression is a constant, variable, function or a combination of these separated by operators and parentheses, written to comply with the rules for constructing the particular type of instruction. An arithmetic expression has numerical value; its type is determined by the type of the operands.

† In an Input/Output list, the name of a dimensioned array implies the entire array rather than the first element.

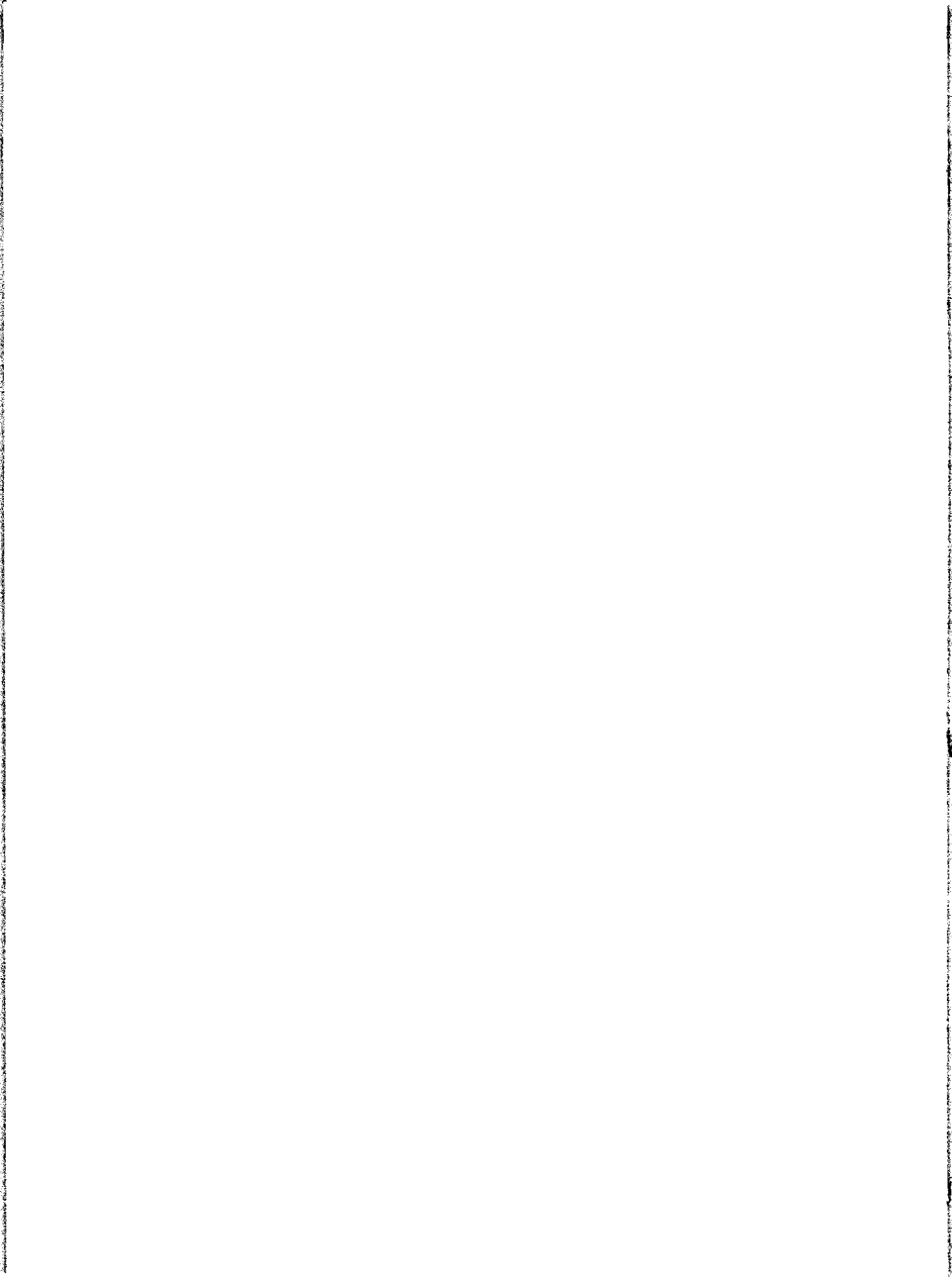
Examples:

A+B-C	. 4+SIN(ALPHA)
X*COS(Y)	A/B+C-D*F
RALPH-ALPH	4+2*IABS(LITE)

2.6 STATEMENTS

Statements are the basic functional units of the language. Executable statements specify actions; non-executable statements describe the characteristics and arrangement of data, editing information, statement functions, and classification of program units.

A statement may be given a numeric label of up to four digits (1 to 9999); a label allows other statements to refer to a statement. Each statement label used must be unique within the program.



Order of Evaluation

In general, the hierarchy of arithmetic operation is:

**	exponentiation	class 1
/	division	class 2
*	multiplication	
-	subtraction	
+	addition	class 3

In an expression with no parentheses or within a pair of parentheses, evaluation basically proceeds from left to right, or in the above order if adjacent operators are in a different class. †

Expressions enclosed in parentheses and function references are evaluated as they are encountered from left to right.

Examples:

In the examples below, s_1, s_2, \dots, s_n indicate intermediate results during the evaluation of the expression; the symbol \rightarrow can be interpreted as "goes to".

- a) Evaluation of class 1 precedes class 3

$A+B**C-D$
 $B**C \rightarrow s_1$
 $s_1+A \rightarrow s_2$
 $s_2-D \rightarrow s_3$ s_3 is the evaluated expression

- b) Evaluation of class 2 precedes class 3

$A*B*C/D+E*F-G/H$
 $A*B \rightarrow s_1$
 $s_1*C \rightarrow s_2$
 $s_2/D \rightarrow s_3$
 $E*F \rightarrow s_4$
 $s_4 + s_3 \rightarrow s_5$
 $G/H \rightarrow s_6$
 $-s_6 \rightarrow s_7$
 $s_7 + s_5 \rightarrow s_8$ s_8 is the evaluated expression

† When writing an integer expression it is important to remember not only the left to right scanning process, but also that dividing an integer quantity by an integer quantity yields a truncated result; thus $11/3 = 3$. The expression $I*J/K$ may yield a different result than the expression $J/K*I$. For example, $4*3/2 = 6$; but $3/2*4 = 4$.

- c) Evaluation of an expression including a function is performed.

$$A+B**C+D+\text{COS}(E)$$

$$B**C \rightarrow s_1$$

$$A+s_1 \rightarrow s_2$$

$$s_2 + D \rightarrow s_3$$

$$\text{COS}(E) \rightarrow s_4$$

$$s_4 + s_3 \rightarrow s_5 \quad s_5 \text{ is the evaluated expression}$$

- d) Parentheses can control the order of evaluation

$$A*B/C+D$$

$$A*B \rightarrow s_1$$

$$s_1 / C \rightarrow s_2$$

$$s_2 + D \rightarrow s_3 \quad s_3 \text{ is the evaluated expression}$$

$$A*B/(C+D)$$

$$A*B \rightarrow s_1$$

$$C+D \rightarrow s_2$$

$$s_1 / s_2 \rightarrow s_3 \quad s_3 \text{ is the evaluated expression}$$

- e) If more than one pair of parentheses or if an exponential expression appears, evaluation is performed left to right.

$$A+B**C-(D*E+F)+(G-H*P)$$

$$B**C \rightarrow s_1$$

$$s_1 + A \rightarrow s_2$$

$$D*E \rightarrow s_3$$

$$s_3 + F \rightarrow s_4$$

$$-s_4 \rightarrow s_5$$

$$s_5 + s_2 \rightarrow s_6$$

$$H*P \rightarrow s_7$$

$$-s_7 \rightarrow s_8$$

$$s_8 + G \rightarrow s_9$$

$$s_9 + s_6 \rightarrow s_{10} \quad s_{10} \text{ is the evaluated expression}$$

Type of Expression

With the exception of exponentiation and function arguments, all operands within an expression must be of the same type. An expression is either real or integer depending on the type of all of its constituent elements.

If either an integer or real operand is exponentiated by an integer operand, the resultant element is of the same type as that of the operand being exponentiated. If both operands are real, the resultant element is real.

Examples:

```
J**I    integer
A**I    real
A**B    real
```

An integer exponentiated by a real operand is not valid.

3.2 ASSIGNMENT STATEMENTS

An arithmetic assignment statement is of the form:

$$v = e$$

The variable, v , may be simple or subscripted; e is an expression. Execution of this statement causes the evaluation of the expression, e , and the assignment of the value to the variable.

Type of Statement

The processing of the evaluated expression is performed according to the following table:

Type of v	Type of e	Assignment rule
Integer	Integer	Transmit e to v without change.
Integer	Real	Truncate and transfer as integer to v .
Real	Integer	Transform integer form of e to floating decimal and transfer to v .
Real	Real	Transmit e to v without change.

Examples:

PC CHANNEL									
C	(Label)	6	8	7	10	15	20	25	30
					A=B**C+D+COS(E)				
					SAM(6)=R-S(6,2)*(T/U)				
					N=W+3.*(X**Y-Z)				
					BAKE=R-I*(J+K*(L-M/N))				
					N=IZZY+LAKE/MOD				

Transmit without change
Transmit without change
Truncate
Convert to real
Transmit without change

3.3 MASKING OPERATIONS

In HP FORTRAN, masking operations may be performed using the Basic External Functions IAND, IOR, and NOT (see Chapter 6). These functions are as follows:

IAND	Form the bit-by-bit logical product of two operands
IOR	Form the bit-by-bit logical sum of two operands
NOT	Complement the operand

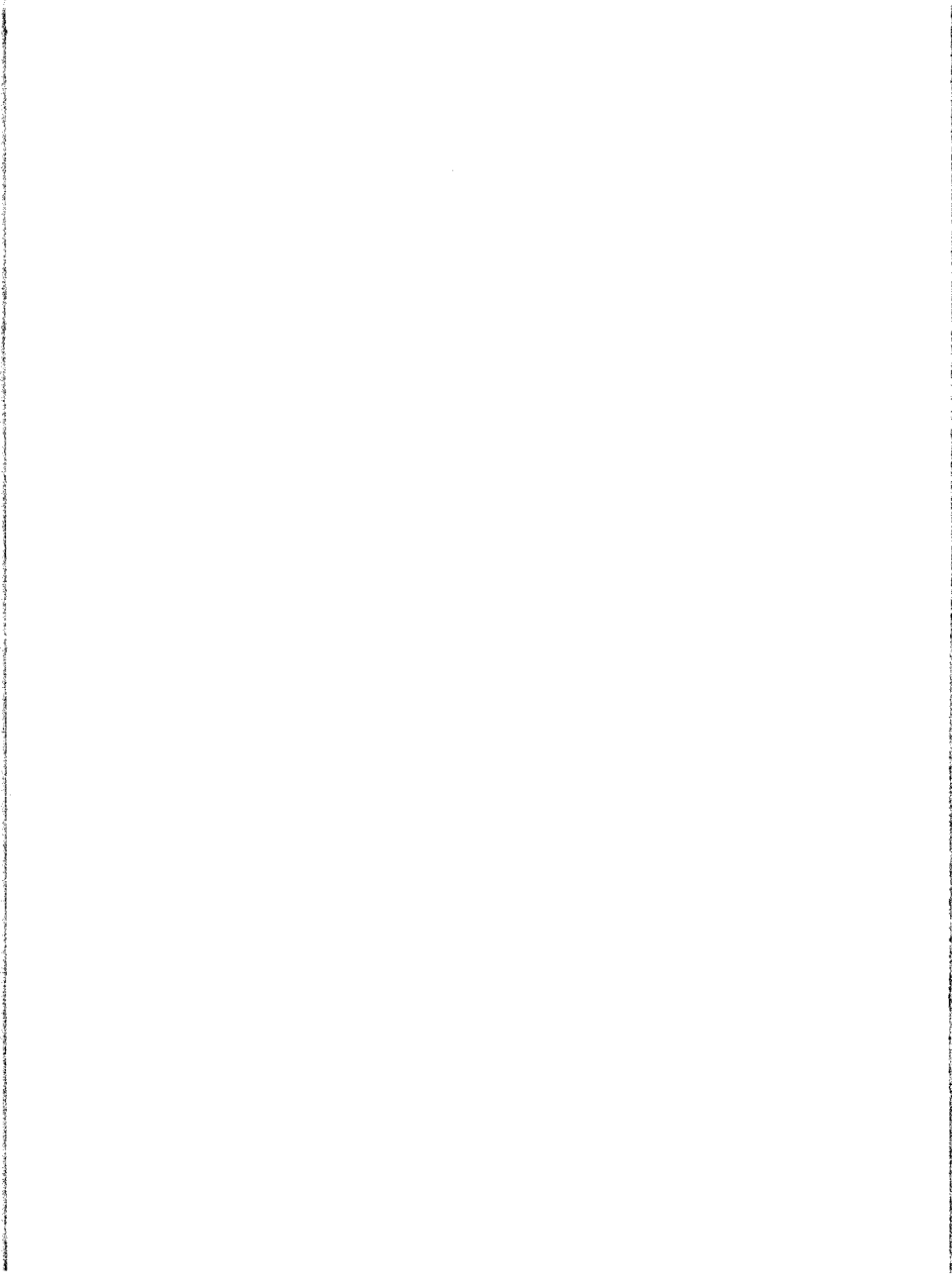
The operations are described by the following table:

Value of Arguments		Value of Function		
a_1	a_2	LAND (a_1, a_2)	IOR (a_1, a_2)	NOT (a_1)
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Examples:

PROJECT NAME: _____										DATE: _____										PROGRAM: _____									
C. Length										STATEMENT																			
0	5	10	15	20	25	30	35	40	45	50																			
IA = 725078																													
IB = 715508																													

LAND (IA, IB) is 70500B
IOR (IA, IB) is 73557B
NOT (IA) is 105270B



The Specifications statements, which include **DIMENSION**, **COMMON**, and **EQUIVALENCE**, define characteristics and arrangement of the data to be processed. These statements are non-executable; they do not produce machine instructions in the object program. The statements must all appear before the first executable statement in the following order: **DIMENSION**, **COMMON**, and **EQUIVALENCE**.

4.1 DIMENSION

The **DIMENSION** statement reserves storage for one or more arrays.

DIMENSION $v_1(i_1), v_2(i_2), \dots, v_n(i_n)$

An array declarator, $v_j(i_j)$; defines the name of an array, v_j , and its associated dimensions, (i_j) . The declarator subscript, i , may be an integer constant or two integer constants separated by a comma. The magnitude of the values given for the subscripts indicates the maximum value that the subscript may attain in any reference to the array.

The number of computer words reserved for a given array is determined by the product of the subscripts and the type of the array name. For integer arrays, the number of words equals the number of elements in the array. For real arrays, two words are used for each element; the storage area is twice the product of the subscripts.

A diagnostic message is printed if an array size exceeds $2^{15} - 1$ locations.

Examples:

DIMENSION SAM (5, 10), ROGER (10, 10), NILE (5, 20)	
Area reserved for SAM	$5 \times 10 \times 2 = 100$ words
Area reserved for ROGER	$10 \times 10 \times 2 = 200$ words
Area reserved for NILE	$5 \times 20 \times 1 = 100$ words

4.2 COMMON

The COMMON statement reserves a block of storage that can be referenced by the main program and one or more subprograms. The areas of common information are specified by the statement form:

$$\text{COMMON } a_1, a_2, \dots, a_n$$

Each area element, a_i , identifies a segment of the block for the subprogram in which the COMMON statement appears. The area elements may be simple variable identifiers, array names, or array declarators (dimensioned array names).

If dimensions for an array appear both in a COMMON statement and a DIMENSION statement, those in the DIMENSION statement will be used.

Any number of COMMON statements may appear in a subprogram section (preceding the first executable statement). The order of the arrays in common storage is determined by the order of the COMMON statements and the order of the area elements within the statements. All elements are stored contiguously in one block.

At the beginning of program execution, the contents of the common block are undefined; the data may be stored in the block by input/output or assignment statements.

Examples:

```
COMMON I (5), A (6), B (4)
```

```
Area reserved for I = 5 words
```

```
Area reserved for A = 12 words
```

```
Area reserved for B = 8 words
```

```
Common area          25 words
```

	Common Block
Origin	I (1)
	I (2)
	I (3)
	I (4)
	I (5)
	A (1)
	A (1)

A (2)
 A (2)
 A (3)
 A (3)
 A (4)
 A (4)
 A (5)
 A (5)
 A (6)
 A (6)
 B (1)
 B (1)
 B (2)
 B (2)
 B (3)
 B (3)
 B (4)
 B (4)

Correspondence of Common Blocks

Each subprogram that uses the common block must include a COMMON statement. Each subprogram may assign different variable and array names, and different array dimensions, however, if corresponding quantities are to agree, the types should be the same for corresponding positions in the block.

Examples:

MAIN PROG COMMON I (5), A (6), B (4)

⋮

SUBPROG1 COMMON J (3), K (2), C (5), D (5)

<u>MAIN PROG</u> <u>reference</u>	<u>Common</u> <u>Block</u>	<u>SUBPROG1</u> <u>reference</u>
I (1)	integer 1	J (1)
I (2)	integer 2	J (2)
I (3)	integer 3	J (3)
I (4)	integer 4	K (1)
I (5)	integer 5	K (2)
A (1)	real 1	C (1)
A (1)	real 1	C (1)

<u>MAIN PROG reference</u>	<u>Common Block</u>	<u>SUBPROG1 reference</u>
A (2)	real 2	C (2)
A (2)	real 2	C (2)
A (3)	real 3	C (3)
A (3)	real 3	C (3)
A (4)	real 4	C (4)
A (4)	real 4	C (4)
A (5)	real 5	C (5)
A (5)	real 5	C (5)
A (6)	real 6	D (1)
A (6)	real 6	D (1)
B (1)	real 7	D (2)
B (1)	real 7	D (2)
B (2)	real 8	D (3)
B (2)	real 8	D (3)
B (3)	real 9	D (4)
B (3)	real 9	D (4)
B (4)	real 10	D (5)
B (4)	real 10	D (5)

If portions of a common block are not referred to by a particular subprogram, dummy variables may be used to provide correspondence in reserved areas.

Examples:

MAIN PROG COMMON I (5), A (6), B (4)

⋮

SUBPROG2 COMMON J (17), B (4)

<u>MAIN PROG reference</u>	<u>Common Block</u>	<u>SUBPROG2 reference</u>
I (1)	integer 1	J (1)
I (2)	integer 2	J (2)
I (3)	integer 3	J (3)
I (4)	integer 4	J (4)
I (5)	integer 5	J (5)

<u>MAIN PROG reference</u>	<u>Common block</u>	<u>SUBPROG2 reference</u>	
A (1)	real 1	J (6)	
A (1)	real 1	J (7)	J (17) is a dummy array. It is
A (2)	real 2	J (8)	not referenced
A (2)	real 2	J (9)	in SUBPROG 2
A (3)	real 3	J (10)	but provides
A (3)	real 3	J (11)	proper correspondence in
A (4)	real 4	J (12)	reserved areas
A (4)	real 4	J (13)	so that SUB-
A (5)	real 5	J (14)	PROG 2 can refer
A (5)	real 5	J (15)	to array B.
A (6)	real 6	J (16)	
A (6)	real 6	J (17)	
B (1)	real 7	B (1)	
B (1)	real 7	B (1)	
B (2)	real 8	B (2)	
B (2)	real 8	B (2)	
B (3)	real 9	B (3)	
B (3)	real 9	B (3)	
B (4)	real 10	B (4)	
B (4)	real 10	B (4)	

The length of the common block may differ in different subprograms, however, the subprogram (or main program) with the longest common block must be the first to be loaded at execution time.

4.3 EQUIVALENCE

The EQUIVALENCE statement permits sharing of storage by two or more entities. The statement has the form:

EQUIVALENCE (k_1), (k_2), ..., (k_n)

in which each k is a list of the form:

a_1, a_2, \dots, a_m

Each a is either a variable name or a subscripted variable; the subscript of which contains only constants. The number of subscripts must correspond to the number of subscripts for the related array declarator.

All names in the list may be used to represent the same location. If an equivalence is established between elements of two or more arrays, there is a corresponding equivalence between other elements of the arrays; the arrays share some storage locations. The lengths may be different or equal.

Examples:

DIMENSION A (5), B (4)

EQUIVALENCE (A (4), B (2))

<u>Array 1</u> <u>Name</u>	<u>Array 2</u> <u>Name</u>	<u>Quantity</u> <u>Element</u>
A (1)		real 1
		real 1
A (2)		real 2
		real 2
A (3)	B (1)	real 3
		real 3
A (4)	B (2)	real 4
		real 4
A (5)	B (3)	real 5
		real 5
	B (4)	real 6
		real 6

The EQUIVALENCE statement establishes that the names A(4) and B(2) identify the fourth real quantity. The statements also establish a similar correspondence between A(3) and B(1), and A(5) and B(3).

An integer array or variable may be made equivalent to a real array or variable; equivalence may be established between different types. The variables may be with or without subscripts.

The effect of an EQUIVALENCE statement depends on whether or not the variables are assigned to the common block. When two variables or array elements share storage, the symbolic names of the variables or arrays may not both appear in COMMON statements in the same subprogram. The assignment of storage to variables and arrays declared in a COMMON statement is determined on the basis of their type and the array

storage is assigned as follows:

<u>Arrays</u>	<u>Quantities</u>	
I (1)	integer 1	} common block
I (2) K (1)	integer 2	
I (3) K (2)	integer 3	
I (4) K (3)	integer 4	
J (1) K (4)	integer 5	
J (2) K (5)	integer 6	

c) Effect of EQUIVALENCE on the length of the common block:

PROGRAMMER	DATE	PROGRAM
STATEMENT		
1	DIMENSION K (7)	
2	COMMON I (4), J (2)	
3	EQUIVALENCE (J (1), K (4))	

storage is assigned as follows:

<u>Arrays</u>	<u>Quantities</u>	
I (1)	integer 1	} common block
I (2) K (1)	integer 2	
I (3) K (2)	integer 3	
I (4) K (3)	integer 4	
J (1) K (4)	integer 5	
J (2) K (5)	integer 6	
K (6)	integer 7	
K (7)	integer 8	

The value of the subscripts for an array being made equivalent to another array should not be such that the origin of the common block is changed (for example, EQUIVALENCE (I (3), K(4))).

<u>Arrays</u>	<u>Quantities</u>
K (1) ← origin changed	integer 1
origin → I (1) K (2)	integer 2
I (2) K (3)	integer 3
I (3) K (4)	integer 4

Arrays

I (4) K (5)

J (1) K (6)

J (2) K (7)

Quantities

integer 5

integer 6

integer 7

If contradictory EQUIVALENCE relationships are specified, a diagnostic message is printed.

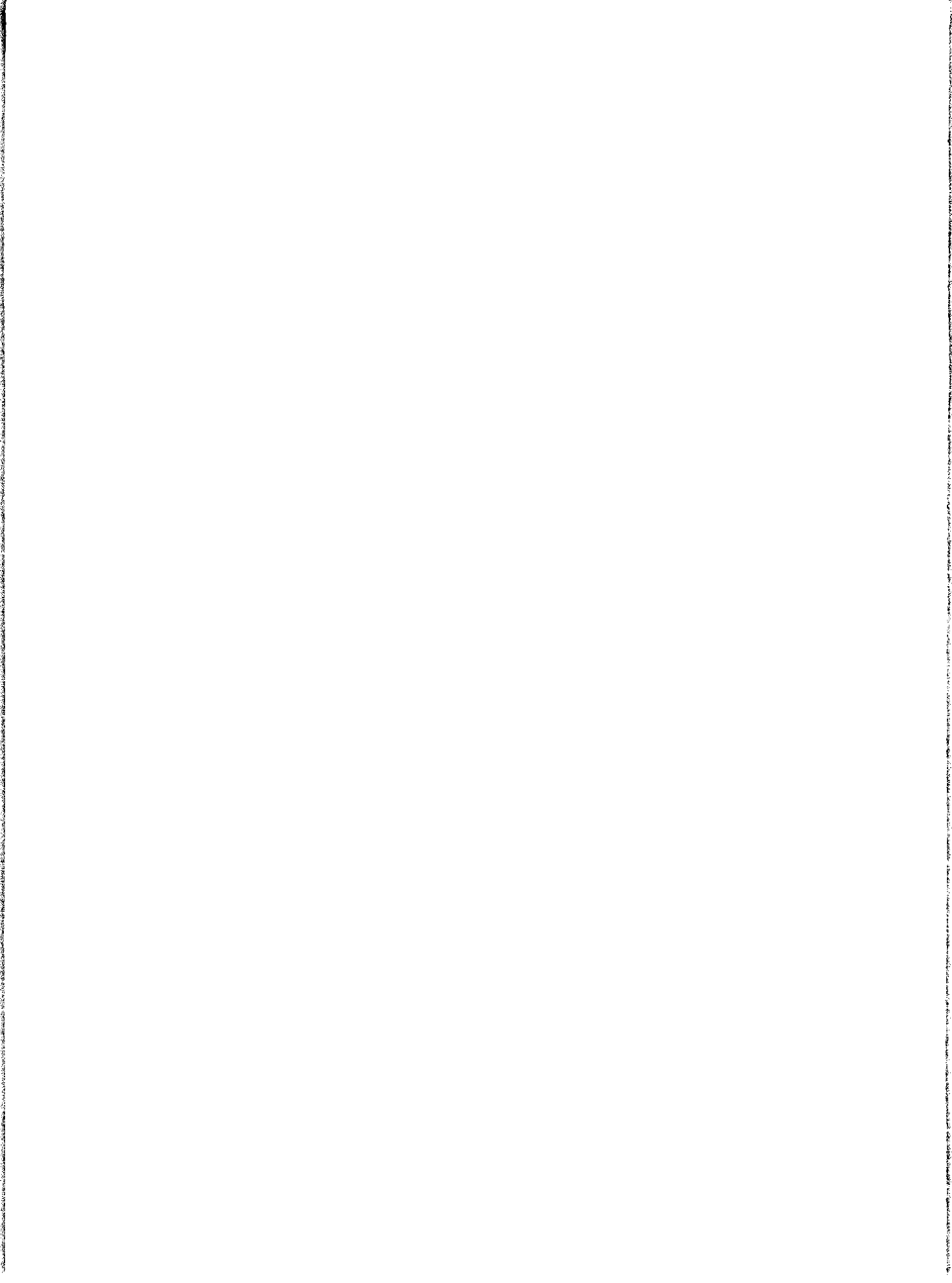
Example:

a)

PROGRAMMER		DATE	PROGRAM									
C	Line#	STATEMENT										
1	5	7	10	15	20	25	30	35	40	45	50	
			EQUIVALENCE (A(2), B(2))									
		*										
		*										
		*										
			EQUIVALENCE (A(5), B(3))									

b)

PROGRAMMER		DATE	PROGRAM									
C	Line#	STATEMENT										
1	5	7	10	15	20	25	30	35	40	45	50	
			EQUIVALENCE (A(2), B(2))									
		*										
		*										
		*										
			EQUIVALENCE (B(3), C(3))									
		*										
		*										
		*										
			EQUIVALENCE (A(5), C(2))									



Program execution normally proceeds from statement to statement as they appear in the program. Control statements can be used to alter this sequence or cause a number of iterations of a program section. Control may be transferred to an executable statement only; a transfer to a non-executable statement will result in a program error which is usually recognized during compilation as a transfer to an undefined label.[†] With the DO statement, a predetermined sequence of instructions can be repeated a number of times with the stepping of a simple integer variable after each iteration.

Statements are labelled by unsigned numbers, 1 through 9999, which can be referred to from other sections of the program. A label up to four digits long precedes the FORTRAN statement and is separated from it by at least one blank or a zero. Imbedded blanks and leading zeros in the label are ignored: 1, 01, 0 1, 0001 are identical.

5.1 GO TO STATEMENTS

GO TO statements provide transfer of control.

GO TO k

This statement, an unconditional GO TO, causes the transfer of control to the statement labelled k .

GO TO (k_1, k_2, \dots, k_n), i

This statement, a computed GO TO, acts as a many-branched transfer. The k's are statement labels and i is a simple integer variable. Execution of this statement causes the statement identified by the label k_j to be executed next, where j

[†]A transfer to a FORMAT statement is not detectable during compilation; if such an error occurs, no diagnostic message is produced.

is the value of i at the time of execution, and $1 \leq j \leq n$. If $i < 1$, a transfer to k_1 occurs; if $i > n$, a transfer to k_n occurs.

Examples:

PROGRAMMER																				DATE										PROGRAM																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
C																				STATEMENT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
Label	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
10	GO	TO	500																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	

At statement 40, control transfers to statement 10, which is an unconditional transfer to statement 500. At 540 control transfers to statement 35.

5.2 IF STATEMENTS

The arithmetic IF statement provides conditional transfer of control

IF (e) k_1 , k_2 , k_3

The e is an arithmetic expression and the k's are statement labels. The arithmetic IF is a three-way branch. Execution of this statement causes evaluation of the expression and transfer of control depending on the following conditions:

$e < 0$, go to k_1
 $e = 0$, go to k_2
 $e > 0$, go to k_3

Examples:

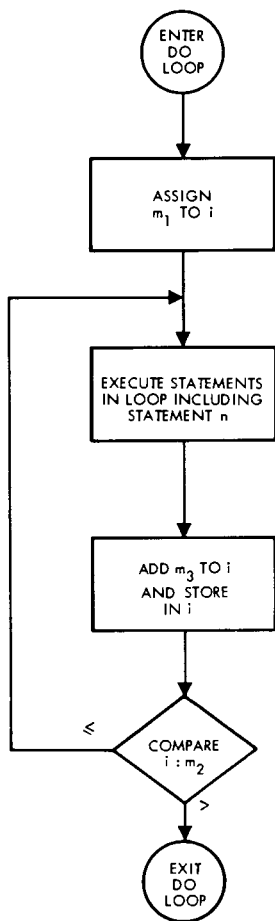
PROGRAMMER																									DATE					PROGRAM																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
C	Label	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			</

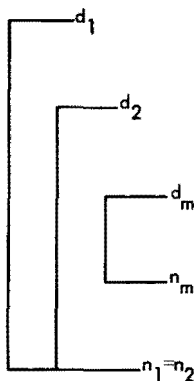
simple integer variables. At time of execution, they all must be greater than zero. If m_3 does not appear (second form), the incrementation value is assumed to be 1.

A DO statement defines a loop. Associated with each DO statement is a range that is defined to be those executable statements following the DO, to and including the terminal statement associated with the DO. At time of execution, the following steps occur:

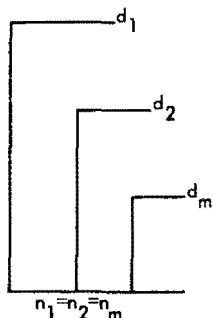
1. The control variable is assigned the value of the initial parameter.
2. The range of the DO is executed.
3. The terminal statement is executed and the control variable is increased by the value of the incrementation parameter.
4. The control variable is compared with the terminal parameter. If less than or equal to the terminal parameter, the sequence is repeated starting at step 2. If the control variable exceeds the terminal parameter, the DO loop is satisfied and control transfers to the statement following n . The control variable becomes undefined.

Should m_1 exceed m_2 on the initial entry to the loop, the range of the DO is executed and control passes to the statement after n . If a transfer out of the DO loop occurs before the DO is satisfied, the current value of the control variable is preserved. The control variable, initial parameters, terminal parameter, and incrementation parameters may not be redefined during the execution of the range of the DO loop.





PROGRAMMER									
C	Label	5	10	15	20	25	30		
I		5 DO	100 I =	1, 20					
J		8 DO	100 J =	1, 10, 3					
K		10 DO	90 K =	1, 20, 2					
		90 CONTINUE							
		100 CONTINUE							



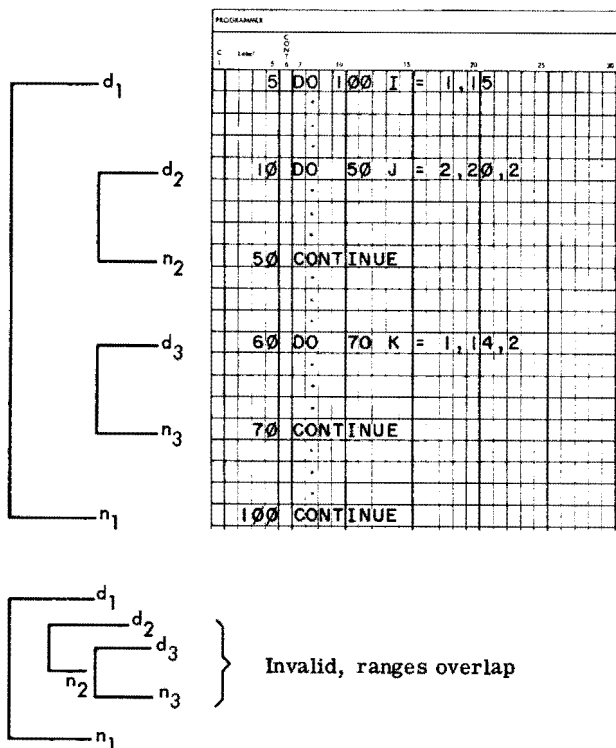
PROGRAMME I

	5	10	15	20	25	30
1	5	DO	100	I = 1, 30, 5		
2		.				
3		.				
4		.				
	10	DO	100	J = 2, 6		
		.				
		.				
		.				
	20	DO	100	K = 5, 50, 5		
		.				
		.				
		.				
	100	CONTINUE				

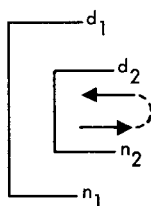
If one or more nested loops have the same terminal statement, when the inner DO is satisfied, the control variable for the next outer loop is incremented and tested against its associated terminal parameter. Control transfers to the statement following the terminal statement only when all related loops are satisfied.

DO loops may be nested in common with other loops as long as their ranges do not overlap.

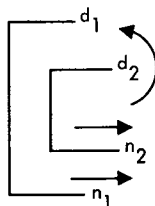
Examples:



In a DO nest, a transfer may be made from an inner loop into an outer loop, and transfer is permissible outside of the loop. It is illegal, however, for a GO TO or IF to initiate a transfer of control from outside of the range of a DO into its range.



VALID
TRANSFERS



INVALID
TRANSFERS

5.4 CONTINUE

This statement acts as no-operation instruction.

CONTINUE

The CONTINUE statement is most frequently used as the last statement of a DO loop to provide a loop termination when a GO TO or IF would normally be the last statement of the loop. If used elsewhere in the source program, it acts as a do-nothing instruction and control passes to the next sequential program statement.

5.5 PAUSE

This statement provides a temporary program halt.

PAUSE n
or
PAUSE

n may be up to four octal digits (without a B suffix) in the range 0 to 7777. This statement halts the execution of the program and types PAUSE on the Standard Teleprinter Output unit. The value of n, if given is displayed in the A-Register. When the RUN button is pressed, program execution resumes at the next statement.

5.6 STOP

The STOP statement terminates the execution of the program.

STOP n
or
STOP

n may be up to four octal digits (without a B suffix) in the range 0 to 7777. This statement halts the execution of the program and types STOP on the Standard Teleprinter Output unit. The value of n, if given, is displayed in the A-Register. If the RUN button is pressed, the halt operation is repeated.

5.7 END

The END statement indicates the physical end of a program or subprogram. It has the form:

END name

The END statement is required for every program or subprogram. The name of the program can be included, but it is ignored by the compiler. The END statement is executable in the sense that it will effect return from a subprogram in the absence of a RETURN statement. An END statement may be labeled and may serve as a junction point.

5.8 END\$

The END\$ statement indicates the physical end of five or less programs or subprograms that are to be compiled at one time. If there are four or less programs, the statement is printed on the source program listing. If there are exactly five, the statement is not printed. If more than five programs are on the same tape, the END\$ may be omitted after the fifth program; the compiler stops accepting input after the fifth is processed.

A FORTRAN program consists of a main program with or without subprograms. Subprograms, which are either functions or subroutines, are sets of statements that may be written and compiled separately from the main program.

The main program calls or references subprograms; and subprograms may call or reference other subprograms as long as the calls are non-recursive. That is, if program A calls subprogram B, subprogram B may not call program A. Furthermore, a program or subprogram may not call itself. A calling program is a main program or subprogram that refers to another subprogram.

In addition to multi-statement function subprograms, a function may be defined by a single statement in the program (statement function) or it may be defined as part of the FORTRAN Library (basic external function). A statement function definition may appear in a main program or subprogram body and is available only to the main program or subprogram containing it. A statement function may contain references to function subprograms, basic external functions, or other previously defined statement functions in the same subprogram. Basic external function references may appear in the main program, subprogram, and statement functions.

Main programs, subprograms, statement functions, and basic external functions communicate by means of arguments (parameters). The arguments appearing in a subroutine call or function reference are actual arguments. The corresponding entities appearing with the subprogram, statement function, or basic external function definition are the dummy arguments.

6.1 ARGUMENT CHARACTERISTICS

Actual and dummy arguments must agree in order, type, and number. If they do not agree in type, errors may result in the program execution, since no conversion takes place and no diagnostic messages are produced.

Within subprograms, dummy arguments may be array names or simple variables; for statement functions, they may be variables only. Dummy arguments are local to the subprogram or statement function containing them and, therefore, may be the same as names appearing elsewhere in the program. A maximum of 63 dummy arguments may be used in a function or subroutine.

No element of a dummy argument list may appear in a COMMON or EQUIVALENCE statement within the subprogram. If it does, a compiler diagnostic results. When a dummy argument represents an array, it should be declared in a DIMENSION statement within the subprogram. If it is not declared, only the first element of the array will be available to the subprogram and the array name must appear in the subprogram without subscripts.

Actual arguments appearing in subroutine calls and function references may be any of the following:

- A constant
- A variable name
- An array element name
- An array name
- Any other arithmetic expression

6.2 MAIN PROGRAM

The first statement of a main program may be the following:

PROGRAM name

The name is an alphanumeric identifier of up to five characters. If the PROGRAM statement is omitted, the compiler assigns the name "FTN."

6.3 SUBROUTINE SUBPROGRAM

An external subroutine is a computational procedure which may return none, one, or more than one value through its arguments or through common storage. No value or type is associated with the name of a subroutine.

The first statement of a subroutine subprogram gives its name and, if relevant, its dummy arguments.

or

The symbolic name, s, is an alphanumeric identifier of up to five characters by which the subroutine is called. If the subroutine is unnamed the compiler will assign the name of "." (period). The a's are the dummy arguments of the subroutine.

The subroutine may define or redefine one or more of its arguments and areas in common so as to effectively return results. It may contain any statements except FUNCTION, another SUBROUTINE statement, or any statement that directly or indirectly references the subroutine being defined. It must have at least one RETURN or END statement which returns control to the calling program.

[illegible]

MUL multiplies the array supplied for MAT by the single value supplied for K to produce values to be stored in array PROD.

6.4 SUBROUTINE CALL

The executable statement in the calling program for referring to a subroutine is:

$$\text{CALL } s(a_1, a_2, \dots, a_n)$$

OR

CALL S

The symbolic name, `s`, identifies the subroutine being called; the `a`'s define the actual arguments. The name may not appear in any specification statements in the calling program.

If an actual argument corresponds to a dummy argument that is defined or redefined in the called subprogram, the actual argument must be a variable name, an array element name, or an array name.

The CALL statement transfers control to the subroutine. Execution of the subroutine results in an association of actual arguments with all appearances of dummy arguments in executable statement and function definition statements. If the actual argument is an expression, the association is by value rather than by name. Following these associations, the statements of the subprogram are executed. When a RETURN or END statement is encountered, control is returned to the next executable statement following the CALL in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until satisfied.

Examples:

RECEIVABLE

DATE

LABOR

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

CALL JIV (15.,12.,ABLE)

COMMON N(10), Q(10)

CALL MUL (I(5,3))

These calls provide actual arguments for the subroutines defined in the previous example. In subroutine JIV, 15. is substituted for P; 12., for W; and ABLE. for H.

For subroutine MUL, the data is passed via COMMON. The value supplied for the dummy argument K is element (5,3) of matrix I of the calling program.

6.5 FUNCTION SUBPROGRAM

A function subprogram is a computational procedure which returns a single value associated with the function name. The type of the function is determined by the name; an integer quantity is returned if the name begins with I, J, K, L, M, or N, otherwise it will be a real quantity.

The first statement of a function subprogram must have the following form:

FUNCTION f (a₁, a₂, ..., a_n)

The symbolic name, f, is an alphanumeric identifier of up to five characters by which the function is referenced. If the function is unnamed the compiler will assign the name of "." (period). The a's are the dummy arguments of the function.

The name of the function must not appear in any non-executable statement in the subprogram. It must be used in the subprogram, however, at least once as any of the following:

- The left-hand identifier of an assignment statement
- An element of an input list
- An actual parameter of a subprogram reference

The value of name at the time of execution of a RETURN or END statement in the subprogram is called the value of the function.

The function subprogram may define or redefine one or more of its arguments and areas in common so as to effectively return results in addition to the value of the function. If the subprogram redefines variables contained in the same expression as the function reference, the evaluation sequence of the expression must be taken into account. Variables in the portion of the expression that is evaluated before the function reference is encountered and the values of variable subscripts are not affected by the execution of the function subprogram. Variables that appear following the function reference are modified according to the subprogram processing.

Examples:

a)

PROGRAMMER									
C	Label	1	5	9	13	17	21	25	29
				FUNCTION		IDIV(I,J)			
				IDIV=I/J					
				RETURN					
				END					

The function IDIV calculates the value of I divided by J. On return to the calling program the result provided is the value of IDIV.

b

```

PROGRAMMER
C      Label  3      7      10      15      20      25
1      * 500
2      FUNCTION IREAD (IUNT)
3      *
4      *
5      *
6      READ(IUNT,*)IREAD
7      *
8      *
9      *
10     RETURN
11     END

```

The function IREAD reads a value from the unit IUNT (specified as an actual parameter in the calling program.) IREAD has this value on return to the calling program.

c'

```

PROGNAME=
C      L      F      IF      XC      ZD
1      .      .      .      .      .
2      FUNCTION SCALL(A,B,C)
3      .
4      .
5      .
6      CALL SUBF(SCALL,A,B,C)
7      .
8      .
9      .
10     RETURN
11     END

```

SCALL is both the function name and an actual parameter of a subroutine call. The value of **SCALL** is provided by **SUBF** and returned to the calling program.

d'

PROGRAMMER										DATE									
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100																			
FUNCTION ZETA(BETA, DELTA, GAMMA)																			
A = BETA**2-DELTA**3																			
GAMMA = A*5.2																			
ZETA = GAMMA**2																			
RETURN																			
END																			

The function defines the value of GAMMA as well as finding the value of ZETA.

6.6 FUNCTION REFERENCE

A function subprogram is referenced by using the name and arguments in an arithmetic expression:

$$f(a_1, a_2, \dots, a_n)$$

The type of function depends on the first letter of the name of the function referenced; the a's are the actual arguments. The reference may appear any place in an expression as an operand. The evaluated function will have a single value associated with the function name. When a function reference is encountered in an expression, control is transferred to the function indicated. Execution of the function results in an association of actual arguments with all appearances of dummy arguments in executable statements and function definition statements. If the actual argument is an expression, this association is by value rather than by name. Following these associations, the statements of the subprogram are executed. When a RETURN or END statement in the function subprogram is encountered, control returns to the statement containing the function reference. During execution the function also may define or redefine one or more of its arguments and areas in common.

Example:

PROGRAM 1		DATE
C	Labels	10 20 30 40 50 60
a)	SANTU=K**IDIV((10,5))+ICON	
b)	SANDU=TAD+IREAD(10B)	

The values of 10 and 5 are provided for I and J: The resulting value of IDIV would be 2. The function IREAD is called with 10B as the unit number. The value of IREAD would be the value of the item read from the device with unit reference number 10g.

c)

PROGRAMMER		DATE	
C	Label	1	2
1	1	2	3
ALPH=BETA*SCALL(10.,9.,8.)			

The actual parameters SCALL are 10., 9., and 8. The value of SCALL would depend on the value supplied by the subroutine SUBF.

d) The program,

PROGRAMMER		DATE		PROGRAM	
C	Label	1	2	3	4
1	1	2	3	4	5
GAMMB=5.0					
RSLT=GAMMB+7.5+ZETA(.2,.3,GAMMB)					

would result in the following calculation:

$$RSLT = 5.0 + 7.5 + ZETA$$

where ZETA would be determined as:

$$\begin{aligned}
 A &= .2**2 - .3**3 = .04 - .027 = .013 \\
 GAMMA &= .013*5.2 = .0676 \text{ (GAMMB is not altered)} \\
 ZETA &= .0676**2 = .00456976 \\
 RSLT &= 5.0 + 7.5 + .00456976 \\
 &= 12.50456976
 \end{aligned}$$

But, the program,

PROGRAMMER		DATE		PROGRAM	
C	Label	1	2	3	4
1	1	2	3	4	5
GAMMB=5.0					
RSLT=ZETA(.2,.3,GAMMB)+7.5+GAMMB					

would result in the following calculations for ZETA and GAMMB:

$$\begin{aligned}
 A &= .2**2 - .3**3 = .04 - .027 = .013 \\
 GAMMA &= .013*5.2 = .0676 = GAMMB \\
 ZETA &= .0676**2 = .00456976 \\
 RSLT &= .00456976 + 7.5 + .0676 \\
 &= 7.57216976
 \end{aligned}$$

When referring to a function which redefines an argument which appears as a variable elsewhere in the same expression, the order of evaluation (i.e., the order in which the expression is stated) is significant.

6.7 STATEMENT FUNCTION

A statement function is defined internally to the program or subprogram in which it is referenced and must precede the first executable statement. The definition is a single statement similar in form to an arithmetic assignment statement.

$$f(a_1, a_2, \dots, a_n) = e$$

The name of the statement function, f , is an alphanumeric identifier; a single value is associated with the name. The dummy arguments, a 's, must be simple variables. One to ten arguments may be used. The expression, e , may be an arithmetic expression and may contain references to basic external functions, previously defined statement functions, or function subprograms. The dummy arguments must appear in the expression. Other variables appearing in the expression have the same values as they have outside the statement function.

The statement function name must not appear in any specification statements in the program or subprogram containing it.

Statement functions must precede the first executable statement of the program or subprogram, but they must follow all specification statements.

A statement function reference has the form:

$$f(a_1, a_2, \dots, a_n)$$

f is the function name and the a 's are the actual arguments. A function reference with its appropriate actual arguments may be used to define the value of an actual argument in a subroutine call or function subprogram reference.

Example:

```

PROGRAMMER
C
Label
1 5 7 10 15 20 25 30
INJR(M,N) = M**2+N**2+5
.
.
.
CALL MATX (INJR(5,2),M)
.
.
SUBROUTINE MATX (J,K)
.
.
.

```

Statement function definition.

Subroutine call using
statement function refer-
ence.

Execution of a statement function reference results in an association of actual argument values with the corresponding dummy arguments in the expression of the function definition, and evaluation of the expression. Following this, the resultant value is made available to the expression that contained the function reference and control is returned to that statement.

Example:

Statement function:

PROGRAMMER															DATE					PROGRAM																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
															STATEMENT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
C	1	5	6	7	10	15	20	23	30	35	40	45	50																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	

Function reference:

PROGRAMMER										DATE										PROGRAM									
										STATEMENT																			
C	1	5	6	7	10	15	20	25	30	35	40	45	50																
C	CALC=RANM+ACES*ABC(7.,11.)																												

6.8 BASIC EXTERNAL FUNCTIONS

Certain basic functions are defined as part of the FORTRAN Library. When one of these appears as an operand in an expression, the compiler generates the appropriate calling sequence within the object program.

The types of these functions and their arguments are defined. The compiler recognizes the basic function and associates the type with the results. The actual arguments must correspond to the type required for the function; if not, a diagnostic message is issued. The functions available are shown below:

Function Name	Definition	Symbolic Name	No. of Arguments	Type of	
				Argument	Function
Absolute Value	$ a $	ABS	1	Real	Real
		IABS	1	Integer	Integer
Float	Conversion from integer to real	FLOAT	1	Integer	Real
Fix	Conversion from real to integer	IFIX	1	Real	Integer
Transfer sign	Sign of a_2 times $ a_1 $	SIGN	2	Real	Real
		ISIGN	2	Integer	Integer
Exponential	e^a	EXP	1	Real	Real
Natural Logarithm	$\log_e(a)$	ALOG	1	Real	Real
Trigonometric Sine	$\sin(a)^\dagger$	SIN	1	Real	Real
Trigonometric Cosine	$\cos(a)^\dagger$	COS	1	Real	Real
Trigonometric Tangent	$\tan(a)^\dagger$	TAN	1	Real	Real
Hyperbolic Tangent	$\tanh(a)$	TANH	1	Real	Real
Square Root	$(a)^{1/2}$	SQRT	1	Real	Real
Arctangent	$\arctan(a)$	ATAN	1	Real	Real
And (Boolean)	$a_1 \wedge a_2$	IAND	2	Integer	Integer
Or (Boolean)	$a_1 \vee a_2$	IOR	2	Integer	Integer
Not (Boolean)	$\neg a$	NOT	1	Integer	Integer
Sense Switch	Sense Switch Register switch (n)	ISSW	1	Integer	Integer

$^\dagger a$ is in radians

Examples:

PROGRAMMER		DATE		PROGRAM	
C		STATEMENTS			
1	2	3	4	5	6
1	SIGND=A,B*C/D-E				
2	SIGNN=ABS(SIGND)				
3	Y=FLOAT(NEWT)				
4	ISGND=I+J*K/L-M				
5	ISGNN=IABS(ISGND)				
6	IAL =JACK*KEN*LARRY				
7	ISAL =ISIGN(IAL,ISGNN)				
8	POWER =EXP(X)				
9	ANTLG=ALOG(Y)				
10	OOHYP=SIN(AGL)				
11	AOHYP=COS(AGL)				
12	OOAH =TANH(AGLH)				
13	HFPR =SQRT(Z)				
14	ARC =ATAN(S)				
15	LPROD=IAND(M,N)				
16	LSUM =IOR(M,N)				
17	LCLMT=NOT(M)				

6.9 RETURN AND END

A subprogram normally contains a RETURN statement that indicates the end of logic flow within the subprogram and returns control to the calling program. It must always contain an END statement.

In function subprograms, control returns to the statement containing the function reference. In subroutine subprograms, control returns to the next executable statement following the CALL. A RETURN statement in the main program is interpreted as a STOP statement.

The END statement marks the physical end of a program, subroutine subprogram, or function subprogram. If the RETURN statement is omitted, END causes a return to the calling program. The END\$ is required in addition to END statements when five or less subprograms are being compiled at one time.

Data transmission between internal storage and external equipment requires an input/output statement and, for ASCII character strings, either a **FORMAT** statement or format control symbols with the input data. The input/output statement specifies the input/output process, such as **READ** or **WRITE**; the unit of equipment on which the process is performed; and the list of data items to be moved. The **FORMAT** statements or control symbols provide conversion and editing information between the internal representation and the external character strings. If the data is in the form of strings of binary values, format control is unnecessary.

7.1 INPUT/OUTPUT LISTS

The input list specifies the names of the variables and array elements to which values are assigned on input. The output list specifies the references to the variables, array elements, and constants whose values are transmitted. The input and output lists are of the same form. The list elements consist of variable names, array elements, and array names separated by commas. The order in which the elements appear in the list is the sequence of transmission. If **FORMAT** statements are used, the order of the list elements must correspond to the order of the format descriptions for the data items. In array elements buffer length is limited to a maximum output of 60 computer words.

Subscripts in an input/output list may be of the form $(\text{exp}_1, \text{exp}_2)$, where exp_1 is one of the following:

$c*v+k$	$v-k$
$c*v-k$	v
$c*v$	k
$v+k$	

where c and k are integer constants and v is a simple integer variable previously defined or defined within an implied **DO** loop.

DO-Implied Lists

A DO-implied list consists of one or more list elements and indexing parameters. The general form is

$$(\dots(\text{list}, i = m_1, m_2, m_3)\dots)$$

- | | |
|------|--|
| list | Any series of arrays, array elements, or variables separated by commas |
| i | Control variable |
| m's | Index parameters in the form of unsigned integer constants or predefined integer variables |

Data defined by the list elements is transmitted starting at the value of m_1 in increments of m_3 until m_2 is exceeded. If m_3 is omitted it is assumed to be one.

An implied DO loop may be used to transmit a simple variable or a sequence of variables more than one time.

Two-dimensional arrays may appear in the list with values specified for the range of the subscripts in an implied DO loop. The general form for an array is:

$$((a(d_1, d_2), i_1 = m_1, m_2, m_3), i_2 = n_1, n_2, n_3)$$

where,

- | | |
|------------|---|
| a | An array name |
| d_1, d_2 | Subscripts of the array in one of the preceding forms |
| i_1, i_2 | Control variables representing either of the variable subscripts d_1 and d_2 |
| m's, n's | Index parameters in the form of unsigned integer constants or predefined integer variables. If m_3 or n_3 is omitted, it is construed as 1. |

The input/output list may contain nested implied DO loops. During execution, the control variables are assigned the values of the initial parameters ($i_1 = m_1$, $i_2 = n_1$). The first control variable defined in the list is incremented first. When the first control variable reaches the maximum value, it is reset; the next control variable to the right is incremented and the process is repeated until the last control variable has been incremented.

If the name of a dimensioned array appears in a list without subscripts, the entire array is transmitted.

Examples:

- a) The DO-implied list:
 $((A(I, J), I=1, 20, 2), J=1, 50, 5)$
replaces the following:
DO x J=1, 50, 5
DO x I=1, 20, 2
transmit A (I, J)
x CONTINUE
- b) Other implied DO loops might be:
 $((ABLE(5*KID-3, 100*LID), KID=1, 100), LID=1, 10)$
 $((A(I, J), I=1, 5), J=1, 5)$ Transmit elements by column
 $((A(I, J), J=1, 5), I=1, 5)$ Transmit elements by row.
- c) Nested implied DO loops:
 $((((A(I, J), B(K, L), K=1, 10), L=1, 15), I=1, 20), J=1, 25)$
 $((A(I, J), B(K), K=1, 10), I=20, 100, 10), K=9, 90, 10)$
- d) Simple variable transmission:
(A, K=1, 10) Transmits 10 values of A.
- e) Dimensioned array transmission:
DIMENSION A(50, 20)

... A ... list element
is equivalent to:
DO x I = 1, 20
DO x J = 1, 50
transmit A(J, I)
x CONTINUE

7.2 FORMAT STATEMENT

ASCII input/output statements may refer to a **FORMAT** statement which contains the specifications relating to the internal-external structure of the corresponding input/output list elements.

FORMAT (spec₁, ..., r(spec_m, ...), spec_n, ...)

The spec's are format specifications and **r** is an optional repetition factor which must be an unsigned integer constant. **FORMAT** specifications may be nested to a depth of one level. The **FORMAT** statement is non-executable and may appear anywhere in the program.

7.3 FORMAT STATEMENT CONVERSION SPECIFICATIONS

The data elements in the input/output lists may be converted from external to internal and from internal to external representation according to **FORMAT** conversion specifications.[†] **FORMAT** statements may also contain editing codes.

Conversion Specifications

rEw.d	Real number with exponent
rFw.d	Real number without exponent
rIw	Decimal integer
r@w {	
rKw {	Octal integer
rAw	Alphanumeric character

Editing Specification

nX	Blank field descriptor
nHh ₁ h ₂ ... h _n }	Heading and labeling descriptors
r" h ₁ h ₂ ... h _n "	
r/	Begin new record

[†] If the type of a variable in the input/output list does not correspond to the type specified in the **FORMAT** statement, the compiler insures that the proper conversion from one type to the other will take place.

Both *w* and *n* are nonzero integer constants representing the width of the field in the external character string; *n* may be omitted if the width is one. *d* is an integer constant representing the number of digits in the fractional part of the string. *r*, the repeat count, is an optional nonzero integer constant indicating the number of times to repeat the succeeding basic field descriptor. Each *h* is one character.

Ew.d Output

The E specification converts numbers in storage to character form for output. The field occupies w positions in the output record; the number appears in floating point form right justified in the field as:

$$\Delta, x_1 \dots x_d \vdash E \pm ee^\dagger$$

$x_1 \dots x_d$ are the most significant digits of the value of the data to be output. ee are the digits in the exponent. Field w must be wide enough to contain significant digits, signs, decimal point, E , and exponent. Generally, w should be greater than or equal to $d + 4$.

If the field is not long enough to contain the output value, an attempt is made to adjust the value of *d* (i.e., truncating part or all of the fraction) so that a number is written in the field. If the remaining value is still too large for the field, dollar signs (\$) are inserted in the entire field. If the field is longer than the output value, the quantity is right-justified with spaces to the left.

Examples:

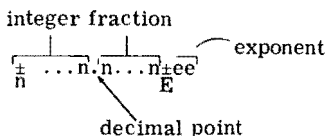
PROGRAMMER									
6	Label	3	7	10	13	26			
					WRITE(4,5)A		A contains	+12.34	or -12.34
	5				FORMAT(E10.3)		Result is	^^.123E+02	or ^-.123E+02
					WRITE(4,5)A		A contains	+12.34	or -12.34
	5				FORMAT(E12.3)		Result is	^^^^.123E+02	or ^^^-.123E+02
					WRITE(4,5)A		A contains	+12.34	or -12.34
	5				FORMAT(E7.3)		Result is	.12E+02	or -.1E+02
					WRITE(4,5)A		A contains	+12.34	
	5				FORMAT(E5.1)		Result is	\$\$\$\$\$	

[†]The caret symbol, \wedge , indicates the presence of a space.

Ew.d Input

The E specification converts the number in the input field (specified by w) to a real number and stores it in the appropriate storage locations.

The input field may consist of integer, fraction, and exponent subfields:



The integer subfield begins with a + or - sign, or a digit and may contain a string of digits terminated by a decimal point, an E , + , - , or the end of the input field.

The fraction subfield begins with a decimal point and may contain a string of digits terminated by an E , + , - , or the end of the input field.

The exponent field may begin with a sign or an E and contains a string of digits. When it begins with E , the + is optional between E and the string. The value of the string of digits should not exceed 38. The number may appear in any positions within the field; spaces in the field are ignored.

Examples:

+1.2345E2
123.456+9
-0.1234-6
.12345E-3
1234
+12345
+1234E6

When no decimal point is present in the input quantity, d acts as a negative power of ten scaling factor. The internal representation of the input quantity will be:

(integer subfield) $\times 10^{-d}$ $\times 10^{(\text{exponent subfield})}$

Example:

```
PROGRAMME3
C
O
N
T
1 Label 5 6 7 10 15 20 25
FORMAT(E12.8)
```

Input quantity = $\wedge\wedge\wedge 1234 + 5\wedge\wedge$

Conversion performed: $1234 \times 10^{-8} \times 10^5$

Result: 1.234

If a d value in the specification conflicts with the a decimal point appearing in an input field, the actual decimal point takes precedence.

Example:

PROGRAMMER															
C 1000															
1	5	8	7	16	15	36	25								
				FORMAT (E12.8)											

Input quantity = ^^^^1.234+5

Quantity stored: 1.234×10^5

The field width specified by `w` should always be the same as the width of the input field. When it is not, incorrect data may be read, converted and stored. The value of `w` should include positions for signs, the decimal point, the letter `E`, as well as the digits of the subfields:

Example:

[illegible]

Assuming input data in contiguous fields:

-12.3E1+1234123.46E-3

$$| \leftarrow 7 \text{ --- } * \text{ --- } 5 \quad * \text{ --- } 9 \rightarrow |$$

The fields read would be:

-12.3E1

+1234

123. 46E-3

and converted as:

-123.
1.234
.12346

However, if specifications were:

PROGRAMMER										DATE	PROGRAM
C										STATEMENT	
1	2	3	4	5	6	7	8	9	10	11	12
10	FORMAT	(E7	.	2	,	E4	.	3	,	E7
			.	2)						

The fields read would be:

-12.3E1
+123
4123.46

and converted as:

-123
.123
4123.46

The effects of possible **FORMAT** specification errors such as the above may not be detected by the system.

Examples:

<u>FORMAT Specification</u>	<u>Input Field</u>	<u>Converted Value</u>
E9.2	+1.2345E2	123.45
E9.4	-0.1234-6	-.0000001234
E4.2	1234	12.34

Fw.d Output

The **F** specification converts real numbers in storage to character form for output. The field occupies *w* positions and will appear as a decimal number, right justified in the field.

△x...x.x...x

The x's represent the decimal digits (maximum of 5) of the integer. When the integer is positive on output, the sign is suppressed. If an output field is too short, dollar signs (\$) will be placed in the output record.

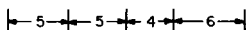
The Iw specification, when used for input, is identical to an Fw.0 specification.

Examples:

PROGRAMMER										DATE									
C																			
1																			
WRITE(6,10)I,J,K,L																			
10 FORMAT(I5,I5,I4,I6)																			

I contains -1234
J contains +12345
K contains +12345
L contains +12345

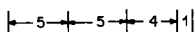
Result: -123412345\$\$\$\$12345



PROGRAMMER										DATE									
C																			
1																			
READ(5,10)I,J,K,L																			
10 FORMAT(I5,I5,I4,I1)																			

Input contains:

-12312123123



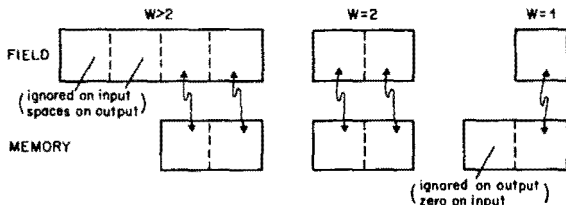
I contains -0123
J contains 12003
K contains 0102
L contains 3

Aw

This specification causes alphanumeric data on an external medium to be translated to or from ASCII form in memory. The associated list element must be of type integer.

On input, if the field, as indicated by w, is greater than 2, the first w-2 characters are ignored; only the last two characters are read. When w equals 2, the two characters are read. If w equals 1, one character is read and stored in the right half of a computer word; zero is entered in the left half.

On output, if the field is greater than 2, two characters are written with right justification in the field; the leading positions are filled with spaces. If w equals 2, the two characters are written. If w equals 1, the character in the right half of the computer word is written.



Example:

```

Input data: AZZ213-ABCXABC137 - ZZ9 (CR) (LF)
DIMENSION ID (5)
READ (5, 10) 12, 11, ID
10  FORMAT (A10, A1, 5A2)

Result: 12 BC
        11 0X
        ID AB
          C1
          37
          -Z
          Z9
  
```

r@w rKw

Octal integer values are converted under either the @ or the K specification. The field is w octal digits in length; the corresponding list element must be of type integer.

On input, if w is greater than or equal to 6, up to six octal digits are stored; non-octal digits appearing within the field are ignored. If the value of the octal digits within the field is greater than 177777, the results are unpredictable. If w is less than 6, or if less than six octal digits are encountered in the field, the number is right justified in the computer word with zero fill on the left.

On output, if the field is greater than 6, six octal digits are written with right justification in the field; the leading positions are filled with spaces. If w equals 6, the six octal digits are written. If w is less than 6, the w least significant octal digits are written.

Example:

```

Input data: 123456-1234562342342342, 396E-05 CR LF
DIMENSION ID(2), IE(2)
READ (5, 10) IB, IC, ID, IE
10  FORMAT (@6, @7, 2@5, 2@4)
  
```

Result: IB 123456
 IC 123456
 ID 023423
 042342
 IE 000036
 000005

nX

The X specification may be used to include n blanks in an output record or to skip n characters on input to permit spacing of input/output quantities. In the specifications list, the comma following X is optional. ^X is interpreted as 1X. 0X is not permitted.

Examples:

OGRAMMER	DATE	PROGRAM
STATEMENT		
Line 1	5	10
WRITE(6,10)A,B,I	15	20
10 FORMAT(E8.3,5X,F6.2,5X,I4)	25	30
	35	40
	45	50

A contains +123.4
 B contains -12.34
 I contains -123

Result: ^1234E2^12.34^123

Input:

WEIGHT^^10^^PRICE^^\$1.98^^TOTAL^^\$19.80

OGRAMMER	DATE	PROGRAM
STATEMENT		
Line 1	5	10
READ(5,10)I,A,B	15	20
10 FORMAT(8X,I2,10XF4.2,10XF5.2)	25	30
	35	40
	45	50

Result: I contains 10
 A contains 1.98
 B contains 19.80

nH₁h₂...h_n

The H specification provides for the transfer of any combination of 8-bit ASCII characters, including blanks. n is an unsigned integer specifying the number of characters to the right of the H that are to be transmitted. The comma following the H specification is optional. ^H is interpreted as 1H. 0H is not permitted.

On output, the ASCII data in the FORMAT statement is written on the unit in the form of comments, titles, and headings.

PROGRAMMER										DATE										PROGRAM									
C Label										STATEMENT																			
1										2										3									
4										5										6									
7										8										9									
10										11										12									
13										14										15									
16										17										18									
19										20										21									
22										23										24									
25										26										27									
28										29										30									
31										32										33									
34										35										36									
37										38										39									
40										41										42									
43										44										45									
46										47										48									
49										50										51									
52										53										54									
55										56										57									
58										59										60									
61										62										63									
64										65										66									
67										68										69									
70										71										72									
73										74										75									
76										77										78									
79										80										81									
82										83										84									
85										86										87									
88										89										90									
91										92										93									
94										95										96									
97										98										99									
100										101										102									
103										104										105									
106										107										108									
109										110										111									
112										113										114									
115										116										117									
118										119										120									
121										122										123									
124										125										126									
127										128										129									
130										131										132									
133										134										135									
136										137										138									
139										140										141									
142										143										144									
145										146										147									
148										149										150									
151										152										153									
154										155										156									
157										158										159									
160										161										162									
163										164										165									
166										167										168									
169										170										171									
172										173										174									
175										176										177									
178										179										180									
181										182										183									
184										185										186									
187										188										189									
190										191										192									
193										194										195									
196										197										198									
199										200										201									
202										203										204									
205										206										207									
208										209										210									
211										212										213									
214										215										216									
217										218										219									
220										221										222									
223										224										225									
226										227										228									
229										230										231									
232										233										234									
235										236										237									
238										239										240									
241										242										243									
244										245										246									
247										248										249									
250										251										252									
253										254										255									
256										257										258									
259										260										261									
262										263										264									
265										266										267									
268</																													

```
WRITE(6,10)I,A,B
10 FORMAT(8HWEIGHT I2,10H PRICE $.F4.2,
C10H TOTAL $.F5.2)
```

Result: WEIGHT 10 PRICE \$1.98 TOTAL \$19.80

Examples:

PROGRAMMER										DATE										PROGRAM									
C										STATEMENT																			
Level																													
1 5 9 13 17 21 25 29 33 37 41 45 49																													
10 READ(5,10)																													
10 FORMAT(31HAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)																													
10 WRITE(6,10)																													

Input: H INPUT ALLOWS VARIABLE HEADERS

Result: H INPUT ALLOWS VARIABLE HEADERS

 $r^{(h_1, h_2, \dots, h_n)}$

This specification also provides for the transfer of any combination of ASCII characters (except the quotation marks). The number of characters transmitted is the number of positions between the two quotation marks; field length is not specified. If *r*, an optional repeat count, is present, the character string within the quotation marks is repeated that number of times. Commas preceding the initial quotation mark and following the closing quotation are optional.

Examples:

PROGRAMMER										DATE										PROGRAM									
C										STATEMENT																			
1	5	10	15	20	25	30	35	40	45	50																			
WRITE(6,10)																													
10 FORMAT(' THIS ALSO IS AN EXAMPLE')																													

Result: THIS ALSO IS AN EXAMPLE

```
WRITE(6,10)
10 FORMAT(3"ABC")
```

Result: ABCABCABC

On input, the number of characters within the quotation marks is skipped on the input field.

New Record

The slash, /, terminates the current record and signals the beginning of a new record of formatted data. It may occur anywhere in the specifications list and need not be separated from the other list elements by commas. Several records may be skipped by indicating consecutive slashes or by preceding the slash with a repetition factor; r-1 records are skipped for r/. On output the slash is used to skiplines, cards, or tape records; on input, it specifies that control passes to the next record or card.

Examples:

[illegible]

Unlimited Groups

FORMAT specifications may be repeated without use of the repetition factor. If list elements remain after all specifications in a FORMAT statement are processed, the rightmost group of repeated (enclosed in parentheses) specifications is used. If there is no repeated group, processing resumes with the first specification in the statement. On output, each time the rightmost parenthesis in the statement, or in the unlimited group, is reached, the current record is terminated.

7.4 FREE FIELD INPUT

By following certain conventions in the preparation of the input data, a 2116A FORTRAN program may be written without use of 'FORMAT' statements. Special symbols included with the ASCII input data items direct the formatting:

space or,	Data item delimiters
/	Record terminator
+ -	Sign of item
. E + -	Floating point number
@	Octal integer
"..."	Comments

All other ASCII non-numeric characters are treated as spaces (and delimiters). Free field input may be used for numeric data only. Free field input is indicated in the FORTRAN READ statement by using an asterisk rather than a number of a FORMAT statement.

Data Item Delimiters

Any contiguous string of numeric and special formatting characters occurring between two commas, a comma and a space or two spaces, is a data item whose value corresponds to a list element. A string of consecutive spaces is equivalent to one space. Two consecutive commas indicate that no data item is supplied for the corresponding list element; the current value of the list element is unchanged. An initial comma causes the first list element to be skipped.

Example:

[illegible]

Input data: 1720, 1966
1980 1492

```
Result: I contains 1720
        J contains 1966
        K contains 1980
        L contains 1492
```

2) PROGRAMME

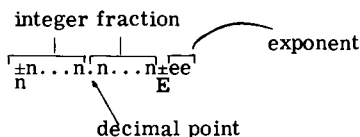
C	Label	5	6	7	10	15	20
					READ(S,*)	I, J, K, L	

Input data: 1266, , 1794, 2000

```
Result:  I contains 1266
         J contains 1966
         K contains 1794
         L contains 2000
```

Floating Point Input

The symbols used to indicate a floating point data item are the same as those used in representing floating point data for **FORMAT** statement directed input:



If the decimal point is not present, it is assumed to follow the last digit.

Examples:

PROGRAMMER										DATE										PROGRAM									
C O N T E N T S										STATEMENT																			
C	Label	5	6	7	10	15	20	25	30	35	40	45	50																
														READ(S,*)A,B,C,D,E															

Input Data: 3.14, 314E-2, 3140-3, .0314+2, .314E1

All are equivalent to 3.14

Octal Input

An octal input item has the following format:

$$@ \mathbf{x}_1 \dots \mathbf{x}_d$$

The symbol @ defines an octal integer. The x's are octal digits each in the range of 0 through 7. List elements corresponding to the octal data items must be type integer.

Record Terminator

A slash within a record causes the next record to be read immediately; the remainder of the current record is skipped.

Example:

PROGRAMMER															DATE										PROGRAM																												
															STATEMENT																																						
C	Line#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50		
																																		</																			

Input data: 987, 654, 321, 123/D₁DESCENDING (CR) (LF)

```
Result: II  contains 987
        JJ  contains 654
        KK  contains 321
        LL  contains 123
        MM contains 456
```

List Terminator

If a line terminates (with a **CR** **LF**) and a slash has not been encountered, the input operation terminates even though all list elements may not have been processed. The current values of remaining elements are unchanged.

Examples:

PROGRAMMER										DATE										PROGRAM									
C O M P I L E R										STATEMENT																			
C	1	5	9	13	17	21	25	29	33	37	41	45	49	53	57														

Input Data:

A=7.987 B=5E2 C=4.6859E-3 (CR) (LF)
J=3456 (CR) (LF)

```
Result:  A contains 7.987
         B contains 5E2
         C contains 4.6859E-3
```

J, X, Y, Z are unchanged.

Comments

All characters appearing between a pair of quotation marks in the same line are considered to be comments and are ignored.

Examples:

"6.7321"	is a comment and ignored
6.7321	is a real number

Input/output statements transfer information between memory and an external unit. The unit is specified as an integer variable that is defined elsewhere in the program or an integer constant.

Each statement may include a list of names of variables, arrays, and array elements. The named elements are assigned values on input and have their values transferred on output.

Records may be formatted or unformatted. A formatted record consists of a string of ASCII characters. The transfer of such a record requires the specification of a **FORMAT** statement or free field input data. An unformatted record consists of a string of binary values.

8.1 UNIT-REFERENCE

The integer specified for an input/output unit is a number which represents a Standard unit assignment or an installation unit assignment. The physical device referenced depends on tables established within the Basic Control System.

The Standard unit numbers are as follows:

<u>Number</u>	<u>Name</u>	<u>Usual Equipment Type</u>
1	Keyboard Input	2752A Teleprinter [†]
2	Teleprinter Output	2752A Teleprinter
3	Program Library	2737A Punched Tape Reader
4	Punch Output	2753A Tape Punch
5	Input	2737A Punched Tape Reader
6	List Output	2752A Teleprinter

[†]If data is to be printed on the Teleprinter as it is read, Unit-Reference number 1 must be used; printing occurs with no other number.

Installation unit numbers may be in the range 7-74₈ with the largest value being determined by the number of units of equipment available at the installation. Each Standard unit may be a separate device, or a single device may be accessed by several Standard unit numbers as well as an installation unit number.[†]

8.2 FORMATTED READ, WRITE

A formatted READ statement is one of the forms:

```
READ (u, f)k  
READ (u, *)k  
READ (u, f)
```

Execution of this statement causes the input of the next ASCII records from unit u. The information is scanned and converted according to the **FORMAT** specification statement, f, and assigned to the elements of list k. If the input is free field, an asterisk is specified in the **READ** statement rather than the label of a **FORMAT** statement. If the list is absent, the **FORMAT** statement should contain editing specifications only.

A formatted **WRITE** statement may have one of the following forms:

```
WRITE (u, f)k  
or  
WRITE (u, f)
```

This statement transfers ASCII information from locations given by names in the list k to output unit u. The values are converted and positioned as specified by the **FORMAT** statement f. If the list is absent, the **FORMAT** statement should contain editing specifications only.

[†]For complete details, see Basic Control System Programmer's Reference Manual.

8.3 UNFORMATTED READ, WRITE

An unformatted READ statement has one of the forms:

```
READ (u)k  
      or  
READ (u)
```

This statement transfers the next binary input record from the unit *u* to the elements of list *k*. The sequence of values required by the list may not exceed the sequence of values from the record. If no list is specified, READ (*u*) skips the next record.

An unformatted WRITE statement has the form:

```
WRITE (u)k
```

Execution of this statement creates the next record on unit *u* from the sequence of values represented by the list *k*.

8.4 AUXILIARY INPUT/OUTPUT STATEMENTS

There are three types of auxiliary input/output statements:

```
REWIND  
BACKSPACE  
ENDFILE
```

A REWIND statement has the form:

```
REWIND u
```

This statement causes the unit *u* to be positioned at its initial point. If the unit is currently at this position, the statement acts as a CONTINUE.

A BACKSPACE statement is as follows:

```
BACKSPACE u
```

BACKSPACE positions the unit *u* so that what had been the preceding record becomes the next record. If the unit is currently at its initial point, the statement acts as a CONTINUE.

An ENDFILE statement is of the form:

ENDFILE *u*

Execution of this statement causes the recording of an end-of-file record on the output unit *u*. If given for an input unit, the statement acts as a CONTINUE.

In addition to the three auxiliary input output statements, a subroutine may be called to perform file and record spacing on magnetic tape. The subroutine call is as follows:

CALL PTAPTE (*u*, *f*, *r*)

u Unit-Reference number of tape device

f File spacing:

 A positive integer specifying the number of files to be spaced forward.

 A negative integer specifying the number of files to be backspaced.

r Record spacing:

 A positive integer specifying the number of records to be spaced forward.

 A negative integer specifying the number of records to be backspaced.

Both file and record spacing may be specified in the same call (e.g., space forward 5 files, then backward 2 records). If file or record spacing is not to be performed, a zero is supplied as the parameter.

If backspacing would result in spacing beyond the Start-of-Tape mark, the spacing operation is terminated and program execution resumes. If forward spacing results in spacing beyond the End-of-Tape marker, the message "*EOT" is printed on the Standard Teleprinter Output unit. When the operator presses RUN, program execution resumes.

The FORTRAN Compiler accepts as input, paper tape containing a control statement and a source language program. The output produced by the Compiler may include a punched paper tape containing the object program; a listing of the source language program with diagnostic messages, if any; and a listing of the object program in assembly level language.

9.1 CONTROL STATEMENT

The control statement must be the first statement of the source program; it directs the compiler.

FTN, P₁, P₂, P₃

FTN is a free field control statement. Following the comma are one to three parameters, in any order, which define the output to be produced. The control statement must be terminated by an end-of-statement mark, **CR** **LF**. Spaces embedded in the statement are ignored.

The parameters may be a combination of the following:

- B Binary output: A program is to be punched in relocatable binary format suitable for loading by the Basic Control System loader.
- L List output: A listing of the source language program is to be produced during Pass One.
- A Assembly listing: A listing of the object program in assembly level language is to be produced in the last pass.

9.2 SOURCE PROGRAM

The source program follows the control statement. Each statement is followed by the end-of-statement mark, **CR** **LF**. Specifications statements must precede executable statements. The last statement in each program submitted for compilation must be an **END** statement. Up to five source programs may be compiled at one time. The last program must be followed by an **END\$** statement, if less than six programs are to be compiled.

The control statement, each of the five programs, and the **END\$** terminator may be submitted on a single tape or on separate tapes. If more than five programs are contained on a tape, the compiler processes the first five and halts with the T-Register containing 102077 (end of Pass 1). The remaining programs must be compiled separately.

9.3 BINARY OUTPUT

The punch output produced by the compiler is a relocatable binary program. It does not include system subroutines introduced by the compiler, or library subroutines referred to in the program.

9.4 LIST OUTPUT

If the List Output parameter is specified, the first 72 characters of each line of the source program is printed on the List Output device. The **END\$** is the last statement printed. If exactly five programs are compiled, however, the **END\$** is omitted from the list.

If the Assembly listing parameter is specified, the program is printed in assembly level language on the List Output device. The program listing is followed by a Symbol Table for the assembly level program.

The format for the assembly level listing is as follows:

<u>Columns</u>	<u>Content</u>
1-5	Zero-relative location (octal) of the instruction
6-7	Blank

9-2 FORTRAN

<u>Columns</u>	<u>Content</u>
8-13	Object code word in octal
14	Relocation or external symbol indicator
15	Blank
16-18	Mnemonic operation code
19	Blank
20-25	Operand address in octal or external symbol name.
26-27	The indicator ",I" if indirect addressing is used.

The Symbol Table listing has the following format:

<u>Columns</u>	<u>Content</u>
1-5	Symbol, statement label, or numeric symbol assigned by the compiler.
6	Blank
7	Relocation indicator
8	Blank
9-14	The zero-relative value of the symbol

The characters that designate an external symbol or type of relocation for the operand address or a symbol in the Symbol Table are:

<u>Character</u>	<u>Relocation Base</u>
Blank	Absolute
R	Program relocatable
X	External symbol
C	Common relocatable

9.5 OPERATING INSTRUCTIONS Paper Tape

The exact operating procedures for a compilation depend on the available hardware configuration.

One possible allocation of equipment might be as follows:

<u>Compiler Input/Output</u>	<u>Standard Unit Designation</u>	<u>Physical Unit Assignment</u>
Binary output	Punch Output	2753A Tape Punch
List output	Teleprinter Output	2752A Teleprinter
Assembly listing	Teleprinter Output	2752A Teleprinter
Source tape(s)	Input	2737A Punched Tape Reader

If there are two output devices as shown above, there are two passes (8K memory) or four passes (4K memory). The list output and an intermediate binary tape are both produced during the first pass; the assembly listing and the binary output are both produced during the last pass.

If one output device is available list output and intermediate binary output are written on the same tape during the first pass (the Compiler ignores the list output when reading the binary data during the second pass). The binary output is then produced in the next to the last pass; and the assembly listing, in the last pass.

The following procedures indicate the sequence of steps for compilation of a source program on paper tape:

- A. Set Teleprinter to LINE and check that all equipment to be used is operable. If the Teleprinter is the only output device, turn ON punch unit.
- B. Load FORTRAN Pass 1 using the Basic Binary Loader:
 1. Place FORTRAN binary tape in the device serving as the Standard Input unit (e.g., Punched Tape Reader).
 2. Set Switch Register to starting address of Basic Binary Loader (e.g., 007700 for 4K memory, 017700 for 8K memory).

3. Press LOAD ADDRESS.
 4. Set Loader switch to ENABLED.
 5. Press PRESET.
 6. Press RUN.
 7. When the computer halts and indicates that the FORTRAN tape is loaded (T-Register contains 102077), set Loader switch to PROTECTED.
- C. If the System Input/Output (SIO) subroutines are on a tape which is separate from FORTRAN Pass 1, load the tape using the Basic Binary Loader as in Step B.
 - D. Set Switch Register to starting address of FORTRAN Pass 1: 000100
 - E. Press LOAD ADDRESS
 - F. Place source language tape in device serving as the Standard Input unit (e.g., Punched Tape Reader).
 - G. Press RUN.
 - H. If more than one source tape, repeat Steps F and G for each tape.
 - I. Perform either of the following depending on memory size:

4K Memory

1. At end of Pass 1 (T-Register contains 102077) load Pass 2 using the Basic Binary Loader as in Step B.
2. Remove binary output from Standard Punch device and place in device serving as the Standard Input unit. (If only one output device, both binary and list output are on the same tape.)
3. Set Switch Register to: 000100
4. Press LOAD ADDRESS
5. Press RUN

6. At end of Pass 2 (T-Register contains 102077), load Pass 3 using the Basic Binary Loader as in Step B.
7. Remove binary output from Standard Punch device and place in device serving as Standard Input unit.
8. Set Switch Register to: 000100
9. Press LOAD ADDRESS.
10. Press RUN.
11. At end of Pass 3 (T-Register contains 102077), load Pass 4 using the Basic Binary Loader as in Step B.
12. Remove binary output from Standard Punch device and place in device serving as Standard Input unit.
13. Set Switch Register to: 000100.
14. Press LOAD ADDRESS.
15. Press RUN.
16. At end of Pass 4, the relocatable binary object tape is on the Standard Punch unit. Either of the following conditions may exist:
 - a. If the T-Register contains 102077, the compilation is complete. If an assembly listing was requested, it is on the List Output Device.
 - b. If the T-Register contains 102001, an assembly listing pass is to be performed:
 - (1) Place binary output from Pass 3 in device serving as Standard Input unit. (Turn off Teleprinter punch unit.)
 - (2) Press RUN.
 - (3) At end of listing pass, T-Register contains 102077.

8K Memory

1. At end of Pass 1 (T-Register contains 102077), load Pass 2 using the Basic Binary Loader as in Step B.
 2. Remove binary output from Standard Punch device and place in device serving as the Standard Input Unit. (If only one output device, both binary and list output are on the same tape.)
 3. Set Switch Register to: 000100
 4. Press LOAD ADDRESS.
 5. Press RUN.
 6. At end of Pass 2, the relocatable binary object tape is on the Standard Punch unit. Either of the following conditions may exist:
 - a. If the T-Register contains 102077, the compilation is complete. If an assembly listing was requested, it is on the List Output device.
 - b. If the T-Register contains 102001, an assembly listing pass is to be performed:
 - (1) Place binary output from Pass 1 in device serving as Standard Input unit. (Turn off Teleprinter punch unit.)
 - (2) Press RUN.
 - (3) At end of listing pass, T-Register contains 102077.
- J. The Basic Control System Loader is used to load the object programs generated by FORTRAN and any referenced library routines. Listed below is a summary of procedures for normal loading of relocatable object programs and library routines (and for the printing of a Memory Allocation Listing): †

† See Section 9.8 for details and options.

1. Load the Basic Control System tape using the Basic Binary Loader as in Step B.
2. Set Switch Register to 000002, press LOAD ADDRESS, and set Switch Register to 000000.
3. Place FORTRAN or Assembler generated relocatable object tape in device serving as Standard Input unit.
4. Press RUN. The loader types "LOAD" when the tape is loaded.
5. If more than one relocatable object tape is to be loaded, repeat Steps J3 and J4 for each. Otherwise, set Switch Register to 000004 to load library routines.
6. Place FORTRAN library tape in device serving as Program Library unit.
7. Press RUN. When the loading operation is complete, the Loader types "*LST". Press RUN to print Loader Symbol Table. When the Loader types "*RUN", the program is ready for execution.
8. Press RUN to initiate execution.

During the operation of the Compiler, the following halts may occur:

<u>T-Register</u>	<u>Explanation</u>	<u>Action</u>
102000	Memory overflow: the program is too long	Irrecoverable error; program must be revised.
102001	End of binary object tape output, start of assembly listing.	If only one output device, place intermediate binary output from previous pass in Standard Input unit and press RUN.

<u>T-Register</u>	<u>Explanation</u>	<u>Action</u>
102007	For all passes except first, unrecognizable record on intermediate binary tape: 1) Punch error on previous pass. 2) Wrong tape supplied as input for pass.	If punch error, restart with Pass 1. If wrong tape, restart current pass: a) Load FORTRAN pass. b) Set Switch Register to 000100. c) Press LOAD ADDRESS. d) Place previous intermediate binary tape in input device. e) Press RUN.
102010	External symbol table overflow: the number of symbols exceeds 255.	Irrecoverable error; program must be revised.
102011	Checksum error on intermediate tape; indicates probable punch error.	Attempt to re-read record (binary records are separated by 4 feed frames). Otherwise, restart with Pass 1.
102066	Tape supply low on 2753A Tape Punch.	Load new tape and press RUN.
102077	Normal end of pass or compilation.	Proceed as indicated in above steps.

For diagnostic messages that might occur during loading see the Basic Control System Programmer's Reference Manual. Diagnostics are also issued by the input/output system provided by FORTRAN and by the FORTRAN library routines (Section 9.9).

9.6 OPERATING INSTRUCTIONS Magnetic Tape

On larger machines with a minimum of 8K memory, the Magnetic Tape Unit may be used to compile FORTRAN programs.

The FORTRAN Compiler operating with magnetic tape uses the scratch area of the system tape for storage of intermediate binary code. Pass 1 writes the intermediate code. At the end of Pass 1, the Inter-Pass Loader is called. This loader searches for and loads Pass 2 of the Compiler. † Control is switched to Pass 2 which spaces forward to the scratch area, processes the intermediate code and produces output on the punch and list devices as requested.

The following procedures indicate the sequence of steps for compilation of a source program on magnetic tape:

- A. Load system tape on Magnetic Tape Unit (See Magnetic Tape System, chapter 8, Operating Manual); check that all equipment to be used is operable.
- B. Load Fortran Pass 1 using Magnetic Tape Binary Loader:
 - 1) Set Switch Register to starting address of Magnetic Tape Binary Loader. For 8K configuration use 017700.
 - 2) Press LOAD ADDRESS.
 - 3) Set FORTRAN identification number in the Switch Register.
 - 4) Press PRESET.
 - 5) Set Loader switch to ENABLED.
 - 6) Press RUN.
 - 7) The computer halts at location 1 with 102077 in the T-Register. The FORTRAN binary tape has been located and loaded.
 - 8) Set Loader switch to PROTECTED.

† The FORTRAN Compiler must be configured prior to use in the magnetic tape system. For complete details see the Standard Software Systems Operating Manual.

- C. Place source language tape in device serving as the Standard Input unit.
- D. Press RUN. †
- E. At the end of compilation, the relocatable binary object tape is on the Standard Punch unit. Either of the following conditions may exist:
 1. If the T-Register contains 102077, the assembly listing was requested and is on the List Output device.
 2. If the T-Register contains 102001, an assembly listing pass is to be performed. Turn off Teleprinter punch unit. Press RUN.

During operation of the tape compiler, the following halts may occur:

<u>T-Register</u>	<u>Explanation</u>	<u>Action</u>
102017	A Write error has occurred during Pass 1.	Restart with beginning of Pass 1.
102027	A Read error has occurred during Pass 2.	Restart with beginning of Pass 1.

† To halt Pass 2 at any time, set Switch 1 up.

9.7 DIAGNOSTIC MESSAGES

Errors detected in the source program are indicated by a numeric code inserted before or after the statement in the List Output.

The format is as follows:

E-eeee: ssss + nnnn
eeee The error diagnostic code shown below.
sss The statement label of the statement in which the error was detected. If unlabeled, 0000 is typed.
nnnn Ordinal number of the erroneous statement following the last labeled statement. (Comment statements are not included in this count.)

<u>Error Code</u>	<u>Description</u>
0001	Statement label error: <ul style="list-style-type: none">a) The label is in positions other than 1-5.b) A character in the label is not numeric.c) The label is not in the range 1-9999.d) The label is doubly defined.e) The label indicated is used in a GO TO, DO, or IF statement or in an I/O operation to name a FORMAT statement, but it does not appear in the label field for any statement in the program (printed after END).
0002	Unrecognized Statement: <ul style="list-style-type: none">a) The statement being processed is not recognized as a valid statement.b) A specifications statement follows an executable statement.c) The specification statements are not in the following order: DIMENSION COMMON EQUIVALENCEd) A statement function precedes a specification statement, or a statement function follows an executable statement.

<u>Error Code</u>	<u>Description</u>
0003	Parenthesis error: There are an unequal number of left and right parentheses in a statement.
0004	Illegal character or format: <ul style="list-style-type: none"> a) A statement contains a character other than A through Z, 0 through 9, or space <code>+=-/(),.\$"</code>. b) A statement does not have the proper format. c) A control statement is missing, misspelled, or does not have the proper format.
0005	Adjacent operators: An arithmetic expression contains adjacent arithmetic operators.
0006	Illegal subscript: A variable name is used both as a simple variable and a subscripted variable.
0007	Doubly defined variable: <ul style="list-style-type: none"> a) A variable name appears more than once in a COMMON statement. b) A variable name appears more than once in a DIMENSION statement. c) A variable name appears more than once as a dummy argument in a statement function. d) A program, subroutine, or function name appears as a dummy parameter; in a specifications statement of the subroutine or function; or as a simple variable in a program or subroutine.
0008	Too many parameters: The dummy parameters for a subroutine or function exceed 63.
0009	Invalid arithmetic expression: <ul style="list-style-type: none"> a) Missing operator b) Illegal replacement
0010	Mixed mode expression: integer constants or variables appear in an arithmetic expression with real constants or variables.

Error Code	Description
0011	Invalid subscript: <ol style="list-style-type: none"> Subscript is not an integer constant, integer variable, or legal subscript expression. There are more than two subscripts (i.e., more than two dimensions). Two subscripts appear for a variable which has been defined with one dimension only.
0012	Invalid constant <ol style="list-style-type: none"> An integer constant is not in the range of -2^{15} to $2^{15}-1$. A real constant is not in the approximate range of 10^{38} to 10^{-38}. A constant contains an illegal character.
0013	Invalid EQUIVALENCE statement: <ol style="list-style-type: none"> Two or more of the variables appearing in an EQUIVALENCE statement are also defined in the COMMON block. The variables contained in an EQUIVALENCE cause the origin of COMMON to be altered. Contradictory equivalence; or equivalence between two or more arrays conflicts with a previously established equivalence.
0014	Table overflow: Too many variables and statement labels appear in the program.
0015	Invalid DO loop: <ol style="list-style-type: none"> The terminal statement of a DO loop does not appear in the program or appears prior to the DO statement. The terminal statement of a nested DO loop is not within the range of the outer DO loop. DO loops are nested more than 10 deep. Last statement in a loop is a GO TO, arithmetic IF, RETURN, STOP, PAUSE, or DO. An indexing parameter is not an unsigned integer constant or simple integer variable or is specified as zero.

Error Code	Description
0016	Statement function name is doubly defined.
0017	A Write error has occurred while producing intermediate code output during Pass 1. (Magnetic tape only.)

9.8 OBJECT PROGRAM LOADING

If absolute binary output was specified, the Basic Binary Loader is used to load the object program tape.

If relocatable binary output was specified, the BCS Relocating Loader is used to load the object program tape. If the program refers to other Assembler or FORTRAN generated object programs, these tapes are loaded by the Relocating Loader at the same time. In general, the FORTRAN Library tape must be submitted for loading also.

Listed below are summaries of procedures for normal loading of object programs:

BASIC BINARY LOADER OPERATING PROCEDURES SUMMARY

- A. Place binary object tape in Standard Input unit.
- B. Set Switch Register to starting address of Basic Binary Loader (e. g., 007700 for 4K memory, 017700 for 8K memory).
- C. Press LOAD ADDRESS.
- D. Set LOADER switch to ENABLED.
- E. Press PRESET.
- F. Press RUN.
- G. When the computer halts with T-Register containing 102077, set LOADER switch to PROTECTED.
- H. Set Switch Register to starting address of object program.
- I. Press LOAD ADDRESS.
- J. Check that all I/O devices are ready and loaded for operation of the program.
- K. Press RUN.

**BASIC CONTROL SYSTEM LOADER
OPERATING PROCEDURES SUMMARY**

- A. Load the Basic Control System tape using the Basic Binary Loader.
- B. Set Switch Register to 000002, press LOAD ADDRESS, and set Switch Register to 000000.
- C. Place Assembler (or FORTRAN) generated relocatable object tape in Standard Input unit.
- D. Press RUN. The loader types "LOAD" if it expects another relocatable or library program.
- E. If more than one relocatable object tape is to be loaded, repeat Steps C and D for each. Otherwise, set Switch Register to 000004 to load library routines.
- F. Place FORTRAN Library tape in device serving as Program Library unit.
- G. Press RUN. When the loading operation is complete, the Loader types "**LST". Press RUN. The Loader types "**RUN" indicating the program is ready for execution. (See the Basic Control System Programmer's Reference manual for error message.)
- H. Press RUN to initiate execution.

9.9 OBJECT PROGRAM DIAGNOSTIC MESSAGES

During execution of the object program, diagnostic messages may be printed on the Teleprinter Output unit by the input/output system supplied for FORTRAN programs. When a halt occurs, the A-Register contains a code which further defines the nature of the error:

<u>Teleprinter Message</u>	<u>A-Register</u>	<u>Explanation</u>	<u>Action</u>
*EQR	Unit Number	Equipment Error: End of input tape on 2752A Teleprinter or 2737A Punched Tape Reader; tape supply low on 2753A Tape Punch. B-Register contains status word of Equipment Table entry.	Place next tape in input device, or for Tape Punch, load new reel of tape. Press RUN.
*FMT	000001	FORMAT error: a) w or d field does not contain proper digits. b) No decimal point after w field c) $w-d \leq 4$ for E specification.	Irrecoverable error; program must be re-compiled.
*FMT	000002	a) FORMAT specifications are nested more than one level deep. b) A FORMAT statement contains more right parentheses than left parentheses.	Irrecoverable error; program must be re-compiled.

<u>Teleprinter Message</u>	<u>A-Register</u>	<u>Explanation</u>	<u>Action</u>
*FMT	000003	a) Illegal character in FORMAT state- ment. b) Format repetition factor of zero. c) FORMAT statement defines more char- acter POSITIONS than possible for device.	Irrecoverable error; program must be re- compiled.
*FMT	000004	Illegal character in fixed field input item or number not right- justified in field.	Verify data.
*FMT	000005	A number has an illegal form (e.g., two E's, two decimal points, two signs, etc.)	Verify data.
*EOT		An attempt has been made to space beyond The End-of-Tape marker on magnetic tape.	Press RUN to resume program execution.

During the execution of an object program referring to the FORTRAN library routines, the following error codes may be printed on the Teleprinter Output unit when error conditions are encountered by the specified subroutine:†

<u>Error Code</u>	<u>Subroutine</u>	<u>Condition</u>
02 UN	ALOG	$a \leq 0$
03 UN	SQRT	$a < 0$
04 UN	.RTOR	$x = 0, y \leq 0$ $x < 0, y \neq 0$
05 OR	SIN, COS	$ a > 2^{14}$
06 UN	.RTOI	$x = 0, i \leq 0$
07 OF	EXP	$ a * \log_2 e \geq 124$
08 OF	.ITOI	i^j out of range
08 UN	.ITOI	$i = 0, j \leq 0$
09 OR	TAN	$ a > 2^{14}$

UN = Floating point underflow

OF = Integer or floating point overflow

OR = Out of range

† For complete details, see FORTRAN Library Routines manual.

ASCII CHARACTER FORMAT

b ₇					0	0	0	0	1	1	1	1	
b ₆					0	0	1	1	0	0	1	1	
b ₅					0	1	0	1	0	1	0	1	
b ₄													
b ₃													
b ₂													
b ₁													
0	0	0	0	0	NULL	DC ₀	␣	0	Ⓟ	P			
0	0	0	0	1	SOM	DC ₁	!	1	A	Q			
0	0	1	0	0	EOA	DC ₂	"	2	B	R			
0	0	1	1	0	EOM	DC ₃	#	3	C	S			
0	1	0	0	0	EOT	DC ₄ (STOP)	\$	4	D	T			
0	1	0	1	0	WRU	ERR	%	5	E	U			
0	1	1	0	0	RU	SYNC	&	6	F	V			
0	1	1	1	0	BELL	LEM	(APOS)	7	G	W			
1	0	0	0	0	FE ₀	S ₀	(8	H	X			
1	0	0	1	0	HT	S ₁)	9	I	Y			
1	0	1	0	0	LF	S ₂	*	:	J	Z			
1	0	1	1	0	V _{TAB}	S ₃	+	,	K	C			
1	1	0	0	0	FF	S ₄	(COMMA)	<	L	\			ACK
1	1	0	1	0	CR	S ₅	-	=	M	J			①
1	1	1	0	0	SO	S ₆	.	>	N	↑			ESC
1	1	1	1	0	SI	S ₇	/	?	O	←			DEL

Standard 7-bit set code positional order and notation are shown below with b₇ the high-order and b₁ the low-order, bit position.

EXAMPLE: The code for "R" is:

b₇ b₆ b₅ b₄ b₃ b₂ b₁
1 0 1 0 0 1 0

LEGEND

NULL	Null/Idle	DC ₁ -DC ₃	Device Control
SOM	Start of message	DC ₄ (Stop)	Device control (stop)
EOA	End of address	ERR	Error
EOM	End of message	SYNC	Synchronous idle
EOT	End of transmission	LEM	Logical end of media
WRU	"Who are you?"	S ₀ -S ₇	Separator (information)
RU	"Are you...?"	b	Word separator (space, normally non-printing)
BELL	Audible signal	<	Less than
FE ₀	Format effector	>	Greater than
HT	Horizontal tabulation	↑	Up arrow (Exponentiation)
SK	Skip (punched card)	←	Left arrow (Implies/Replaced by)
LF	Line feed	\	Reverse slant
V _{TAB}	Vertical tabulation	ACK	Acknowledge
FF	Form feed	Ⓟ	Unassigned control
CR	Carriage return	ESC	Escape
SO	Shift out	DEL	Delete/Idle
SI	Shift in		
DC ₀	Device control reserved for data link escape		

FORTRAN STATEMENTS AND FUNCTIONS B

	Page
Executable Statements	
Assignment	
$v = e$	3-4
Statement Function	
$f(a_1, a_2, \dots, a_n) = e$	6-9
Control	
CALL s	6-4
CALL s (a_1, a_2, \dots, a_n)	6-4
CONTINUE	5-9
DO n i = m_1, m_2, m_3	5-3
END	5-10, 6-12
END\$	5-10, 6-12
GO TO k	5-1
GO TO (k_1, k_2, \dots, k_n), i	5-1
IF (e) k_1, k_2, k_3	5-2
IF (e) k_1, k_2	5-3
PAUSE; PAUSE n	5-9
RETURN	6-12
STOP; STOP n	5-9
Input/Output	
BACKSPACE u	8-3
ENDFILE u	8-3
READ (u)	8-3
READ (u)k	8-3
READ (u, f)	8-2
READ (u, f)k	8-2

READ (u, *)k	8-2
REWIND u	8-3
WRITE (u)k	8-3
WRITE (u, f)	8-2
WRITE (u, f)k	8-2

Nonexecutable Statements

Specification

DIMENSION $v_1(i_1), v_2(i_2), \dots, v_n(i_n)$	4-1
COMMON a_1, a_2, \dots, a_n	4-2
EQUIVALENCE $(k_1), (k_2), \dots, (k_n)$	4-5

Format

FORMAT (spec ₁ , ..., r(spec _m , ...), spec _n , ...)	7-4
---	-----

Subprogram

FUNCTION f (a_1, a_2, \dots, a_n)	6-5
PROGRAM name	6-2
SUBROUTINE s	6-3
SUBROUTINE s (a_1, a_2, \dots, a_n)	6-3

Functions

Function Name	Definition	Symbolic Name	No. of Arguments	Type of	
				Argument	Function
Absolute Value	$ a $	ABS	1	Real	Real
Float	Conversion from integer to real	IABS	1	Integer	Integer
		FLOAT	1	Integer	Real
Fix	Conversion from real to integer	IFIX	1	Real	Integer
Transfer sign	Sign of a_2 times $ a_1 $	SIGN	2	Real	Real
		ISIGN	2	Integer	Integer
Exponential	e^a	EXP	1	Real	Real
Natural Logarithm	$\log_e(a)$	ALOG	1	Real	Real
Trigonometric Sine	$\sin(a)^\dagger$	SIN	1	Real	Real
Trigonometric Cosine	$\cos(a)^\dagger$	COS	1	Real	Real
Trigonometric Tangent	$\tan(a)^\dagger$	TAN	1	Real	Real
Hyperbolic Tangent	$\tanh(a)$	TANH	1	Real	Real
Square Root	$(a)^{1/2}$	SQRT	1	Real	Real
Arctangent	$\arctan(a)$	ATAN	1	Real	Real
And (Boolean)	$a_1 \wedge a_2$	LAND	2	Integer	Integer
Or (Boolean)	$a_1 \vee a_2$	IOR	2	Integer	Integer
Not (Boolean)	$\neg a$	NOT	1	Integer	Integer
Sense Switch	Sense Switch Register Switch (n)	ISSW	1	Integer	Integer

$^\dagger a$ is in radians

A FORTRAN program can refer to a subprogram that has been prepared using Assembler source language. The subprogram may be treated as a subroutine or as a function. The object code programs generated by FORTRAN and by the Assembler are then linked together by the Basic Control System Relocating Loader when the programs are loaded.

FORTRAN REFERENCE

In the FORTRAN program, a subroutine is called using the following statement:

$$\text{CALL } s(a_1, a_2, \dots, a_n)$$

The symbolic name, s , identifies the subroutine and the a 's are the actual arguments.

If the subprogram is a function, it is referenced by using the name and the actual arguments in an arithmetic expression:

$$f(a_1, a_2, \dots, a_n)$$

As a result of either the call or the reference, FORTRAN generates the following coding sequence:

JSB s/i	Transfers control to subroutine or function
DEF* $+n+1$	Defines return location
DEF a_1	Defines address of a_1
DEF a_2	Defines address of a_2
:	
:	
DEF a_n	Defines address of a_n

The words defining the addresses of the arguments may be direct or indirect depending on the actual arguments. For example, an integer constant as an actual argument would yield a direct reference; an integer variable might yield an indirect reference.

If the subprogram being referenced is a subroutine, it may return none, one, or more than one value through its arguments or through common storage. If the subprogram is a function, it is assumed to return a single value in the accumulators: a function of type integer returns a value in the A-Register; a function of type real returns a value in the A- and B-Registers.

The subprogram may transfer values directly by accessing the words in the calling sequence or it may make use of the FORTRAN library subroutine .ENTR to aid in the transfer.

DIRECT TRANSFER OF VALUES

Any suitable technique may be used to obtain or deliver values for the arguments and to return control to the calling program. If address arithmetic is used in conjunction with an argument (e.g., to process elements of an array), the base location must be a direct reference; the location given in the calling sequence must be checked to determine if it is a direct or indirect reference. If it is an indirect reference the location to which it points must also be checked, and so forth.

Example:

PROGRAMMER										DATE	PROGRAM
STATEMENT											
1	2	3	4	5	6	7	8	9	10	11	12
		NAM	AMSUB								
		ENT	AMSUB								
AMSUB		NOP									
		LDA	AMSUB, I								
		STA	RETRN								
NXTAG		ISZ	AMSUB								
		LDA	AMSUB								
		CPA	RETRN								
		JMP	RETRN, I								
PRISAG											
		LDA	AMSUB, I								
		LDA	0, I								
		LCA	AMSUB, I								

	DLD	0,I	VALUE INTO A AND B.
	LDA	AMSUB,I	STORE ONE-WORD VALUE IN ARGUMENT
	STA	OUTAD	LOCATION.
	LDA	W1VAL	
	STA	OUTAD,I	
	LDA	AMSUB,I	STORE TWO-WORD IN ARGUMENT
	STA	OUTAD	LOCATIONS.
	DLD	W2VAL	
	DST	OUTAD,I	
	LDA	AMSUB,I	A CONTAINS ADDR OF LOCATION OF
	SSA		ARGUMENT. TO DETERMINE IF REF IS
	JMP	*+2	INDIRECT, TEST BIT 15. IF ONE,
	JMP	*+5	SET TO ZERO WITH AND, THEN LOAD
	AND	ANMSK	A WITH REFERENCED LOCATION.
	LDA	0,I	REPEAT TEST WITH NEXT REF. WHEN
	JMP	*-5	DIRECT REF ENCOUNTERED, PROCEED
ANMSK	OCT	077777	WITH PROCESSING.
	JMP	NXTAG	RETURN THROUGH HERE WHEN NEXT
RETRN	BSS	1	ARGUMENT IS REQUIRED.
OUTAD	BSS	1	
W1VAL	BSS	1	
W2VAL	BSS	2	
	END		

The preceding example assumes that each argument is processed or partially processed before the next is obtained or delivered. Control returns to the calling program when all arguments have been picked up or delivered.

TRANSFER VIA .ENTR

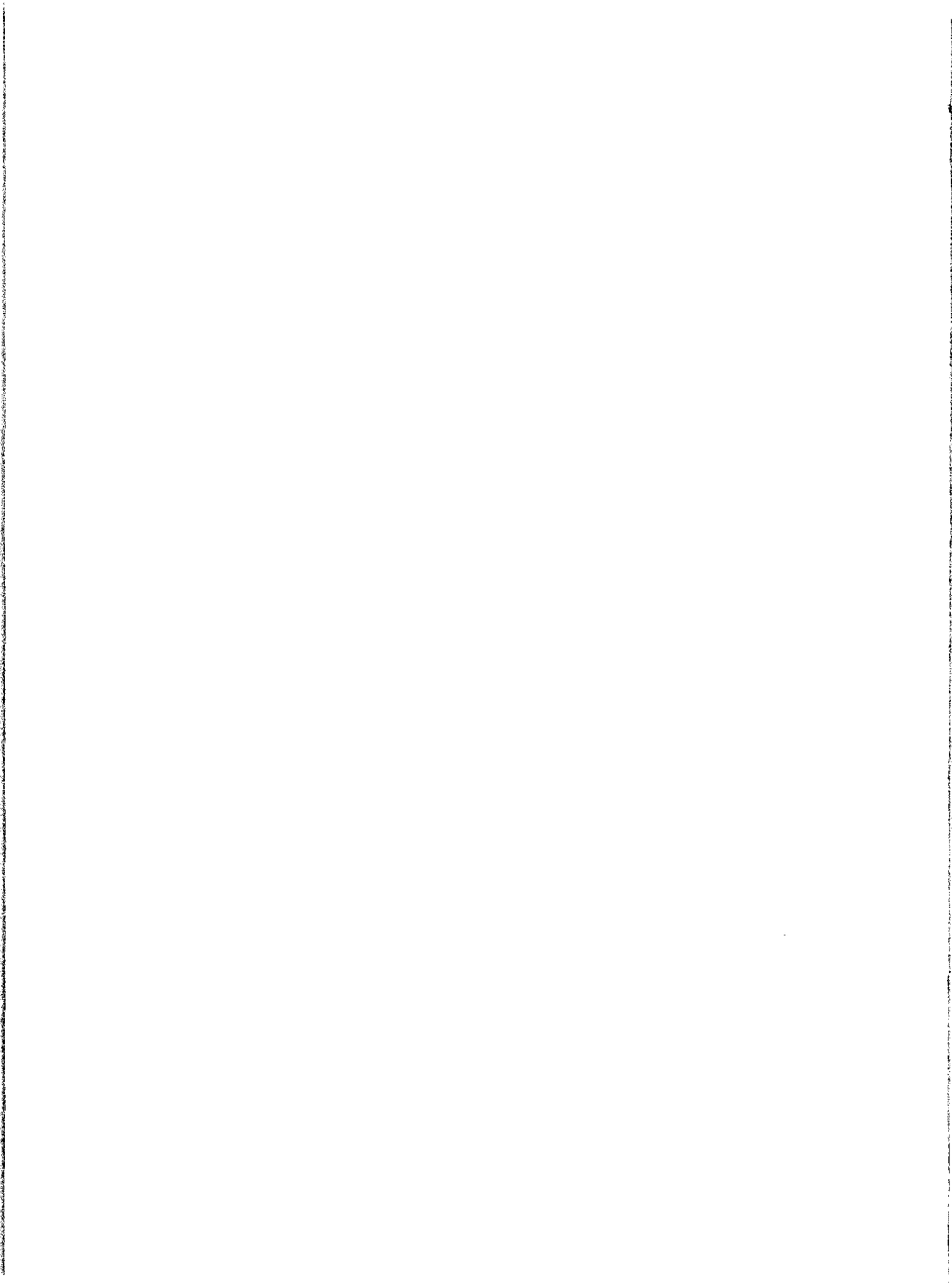
The transfer of values to or from the locations listed in the calling sequence may be facilitated through use of the FORTRAN library subroutine .ENTR. This subroutine moves the addresses of the arguments into an area reserved within the Assembly language subroutine. The addresses stored in the reserved area are all direct references; .ENTR performs all the necessary direct/indirect testing, etc. It also sets the correct return address in the entry point location.

The general form of the subroutine is:

	NAM s	The subroutine name is s.
	ENT s	
	EXT .ENTR	.ENTR must be declared as external.
a	BSS n	Reserves n words of storage for the
s	NOP	addresses of the arguments; this pseudo
		instruction must directly precede the
		entry point location, s.
	JSB .ENTR	
	DEF a	Defines first location of area used to
(First instruction)		store argument addresses.
	.	
	.	
	.	
	JMP s, I	
	END	

Example:

PROGRAMMER		DATE		PROGRAM	
Label		Operation	Comments	Statement	
		NAM AMSUB			
		ENT AMSUB			
		EXT .ENTR			
AGMTS	BSS	5			
AMSUB	NOP				
	JSB .ENTR				
	DEF AGMTS				
PR\$AS			PROCESS ARGUMENTS AS REQUIRED		
	LDA	AGMTS, I	PICK UP VALUE OF FIRST ARGUMENT		
	OLD	AGMTS+1, I	PICK UP VALUE OF SECOND ARGUMENT		
	LDA	WIVAL	STORE VALUE FOR THIRD ARGUMENT		
	STA	AGMTS+2, I			
	OLD	W2VAL	STORE VALUE FOR FOURTH ARGUMENT		
	DST	AGMTS+3, I			
	LDA	AGMTS+4	PICK UP ADDRESS OF FIFTH ARGUMENT		
	JMP	AMSUB, I	RETURN TO CALLING PROGRAM		
WIVAL	BSS	1			
W2VAL	BSS	2			
	END				



It is possible to program and read data from instruments in a FORTRAN program. Pertinent instrument configurations are:

HP 2401C Digital Voltmeter

HP 2401C Digital Voltmeter with a Digital Voltmeter Programmer card.

HP 2401C Digital Voltmeter, Digital Voltmeter Programmer Card, HP 2911A/B Scanner and Scanner Programmer Card.

A. DIGITAL VOLTMETER

In a configuration without a programmer card data can be read in from the voltmeter but it is not possible to program the voltmeter from the computer. Thus the range, function, and gate time must be set using the front panel switches on the instrument. The "SAMPLING RATE" knob must be set so that the internal "hold-off" is removed (max. sampling rate).

WRITE

The WRITE statement removes the "hold-off" on the voltmeter; the instrument is enabled to take a measurement using the range, function, and gate time set on the front panel.

WRITE (u) Where: u is variable name equal to the unit number assigned to the digital voltmeter when preparing the Basic Control System.

READ

The READ statement transfers the measurement result to the computer. Data may be read in as formatted ASCII or as thirty-two bits BCD.

BCD input:

READ (u) i, j or READ (u) x

Where: u as defined above.

i, j Are integer variable names where the thirty-two bits of data will be stored.

x is a floating point variable name where the thirty-two bits of data will be stored.

The digital voltmeter provides the computer with eight BCD digits (32 bits) as follows:

r	f	d ₆	d ₅
d ₄	d ₃	d ₂	d ₁

Where: R = Range

F = Function

D6-D1 = the six digit measurement

The range is a negative power of ten, and the six digits form a number in the range 000000 to 999999. For example, if the range were 4 and the function +DC, and D6-D1 = 163525, the data would be:

4	+DC	1	6
3	5	2	5

$$+163525 \times 10^{-4} \text{ or } 16.3525 \text{ Vdc}$$

The following table shows the function codes for 8-4-2-1 HP 2401C Integrating Digital Voltmeters.

FUNCTION	8-4-2-1 CODE
PERIOD	0-0-0-0
+Vdc	0-0-0-1
-Vdc	0-0-1-0
kHz	0-0-1-1
k-Ω	0-1-0-0
m-Ω	0-1-0-1
Overload	1001
Vac	1011

ASCII input:

READ (u, f) x, i

- Where:
- u as defined above.
 - f label of FORMAT statement defining the format of the reading: (2X, E10.0, I4) or (2X, F10.0, I4).
 - x is the name of a floating point variable which will be set to the six digits of data times ten to negative range.
 - i (this conversion valid only for 8-4-2-1 digital voltmeters) is the name of an integer variable which is set to the function code.

The value of "i" for various function codes is:

I = 0 PERIOD	I = 4 k Ω
I = 1 + Vdc	I = 5 m Ω
I = 2 -Vdc	I = 9 OVERLOAD
I = 3 kHz	I = 11 Vac

The Formatter provides the computer with a string of characters as below:

$\underbrace{r \ f \ d_6 \ d_5 \ d_4 \ d_3 \ d_2 \ d_1 \ E-\theta r}_{x} \ \underbrace{\wedge f_1 \ f_2}_i$

B. DIGITAL VOLTMETER AND A VOLTMETER PROGRAMMER CARD

With the programmer card the voltmeter can be directly programmed by the computer. (The instrument must be set for remote programming.)

WRITE

The following FORTRAN statements will (1) remove the computer "hold-off", (2) program the range, function, and gate time, and (3) send an "encode" command to the digital voltmeter which removes the internal instrument "hold-off".

WRITE (u₁)

WRITE (u₂) code

- Where:
- u₁ is an integer constant or variable name equal to the unit number of the digital voltmeter Data Source Interface card as assigned when the Basic Control System is prepared.

u_2 is an integer constant or variable name equal to the unit number of the digital voltmeter programmer card as assigned when the Basic Control System is prepared.

code is an integer variable name equal to the program code.

The digital voltmeter program code is a sixteen bit quantity. The sign bit (bit 15) will cause an "encode" if it is one, and will suppress the "encode" if it is zero. Bits 14-8 are ignored, and the function of bits 7-0 is illustrated in the table below.

Coding for Voltmeter Range, Function, and Sample Period

PROGRAM CODE BITS 7-0			PROVIDES
7-6	5-4-3	2-1-0	
		000	Autorange
		001	+ 10 Gain, 2411A
		010	0.1V Range
		211	1V Range
		100	10V Range
		101	100V Range
		110	1000V Range
		111	10 Megohm Range
	000		AC Normal
	001		AC Fast
	010		Frequency
	011		Period
	100		DC Volts
	101		Ohms
00			1 Sec. Sample Period
01			0.1 Sec. Sample Period
10			0.01 Sec. Sample Period

READ

The measurement result is transferred to the computer by the following statement.

READ (u), f) x, i

(as explained section A)

Example:

In the following example, assume that the digital voltmeter programmer card has been assigned unit number 10 during the preparation of the Basic Control System, and the DSI card on the corresponding digital voltmeter has been assigned the unit number 11. A measurement will be made using the AC/Ohms Converter:

Range:	10 volts
Function:	AC Normal
Gate Time:	1 second
Encode:	by the Digital Voltmeter Programmer Card

```
C
C REMOVE THE HOLD-OFF ON THE VOLTMETER
C
C WRITE(11)
C
C PROGRAM THE DIGITAL VOLTMETER AND CONVERTER ENCODE THEM
C
C NCODE = 100004B
C WRITE(10), NCODE
C
C READ THE DATA AND THE FUNCTION CODE
C
C READ(11,20), DATA, IFUN
C 20 FORMAT (2X, E10.0, I4)
C
C CHECK FOR OVERLOAD
C
C IF (IFUN-9) 50, 40, 50
C
C WRITE MESSAGE IF OVERLOAD
C
C 40 WRITE(2,41)
C 41 FORMAT ("OVERLOAD ON UNIT # 11")
C
C CONTINUE
C
C 50 CONTINUE
```

C. DIGITAL VOLTMETER, VOLTMETER PROGRAMMER CARD, SCANNER AND SCANNER PROGRAMMER CARD

When the HP 2401C Digital Voltmeter (and HP 2410B AC/Ohms Converter, and HP 2411A Guarded Data Amplifier) is used with the DSI card and the DVM Programmer Card, and the HP 2911A/B Scanner is used with the Scanner Programmer Card, it is possible to program the range, function, and gate time on the voltmeter (and converter and amplifier), and to program the scanner channel number and settling delay. (The instruments must be set for remote programming.)

WRITE

If the scanner is on the correct channel the following format is correct:

WRITE (u_1)

WRITE (u_2) code (as in section B; bit 15 = 0)

If the scanner needs to be set to the correct channel the following FORTRAN statements will (1) remove the computer "hold-off", (2) program the range, function, and gate time on the digital voltmeter (and converter and amplifier), and (3) set the function, channel number, and scanner delay on the scanner, and (4) encode the voltmeter after the scanner delay:

WRITE (u_1)

WRITE (u_2) code

WRITE (u_3) chan, scan

Where: u_1 as defined above

u_2 as defined above

code as defined above

u_3 is an integer constant or variable name equal to the unit number of the scanner programmer as assigned when the Basic Control System is prepared.

chan is variable name equal to the channel number (0-999).

scan is integer variable name equal to the scanner program code.

The scanner program code is a sixteen bit quantity. Only the last four bits are used. This code can be considered a two digit octal number, where the digits have the following significance:

scan = D1D2

D1 (Function)

0 = AC/DC Volts
1 = frequency period
2 = resistance

D2 (Scanner Delay)

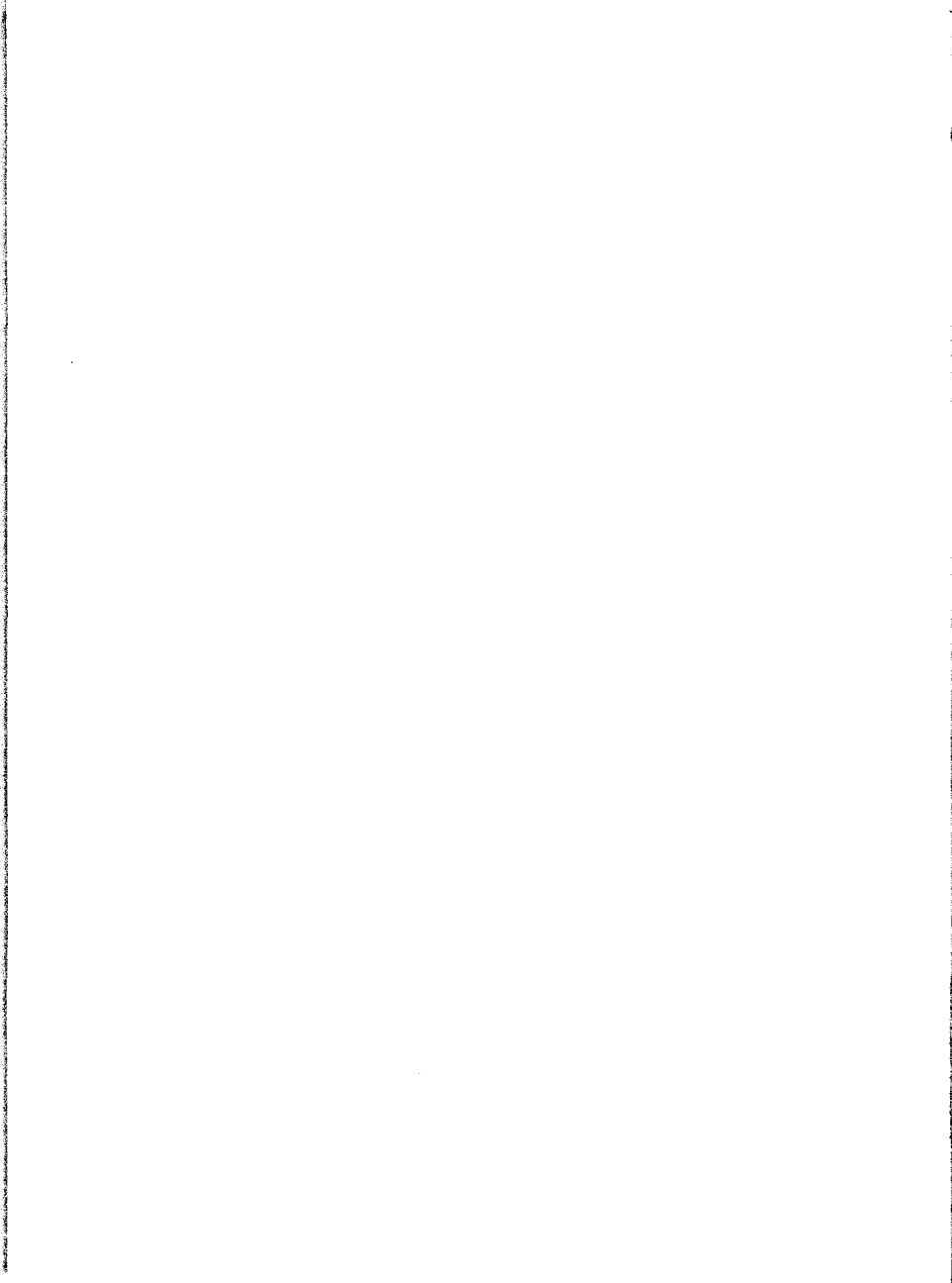
<u>Normal</u>	<u>High Voltage</u>
0 = 15 ms	22 ms
1 = 17.5 ms	27 ms
2 = 22 ms	22 ms
3 = 27 ms	27 ms
4 = 42 ms	145 ms
5 = 62 ms	500 ms
6 = 145 ms	145 ms
7 = 500 ms	500 ms

READ

The measurement result is transferred to the computer by the following statement:

READ (u, f) x

(as explained in section A)



Using Simpson's rule, calculate the value of the integral:

$$\int_a^b \frac{\cos x}{x} dx$$

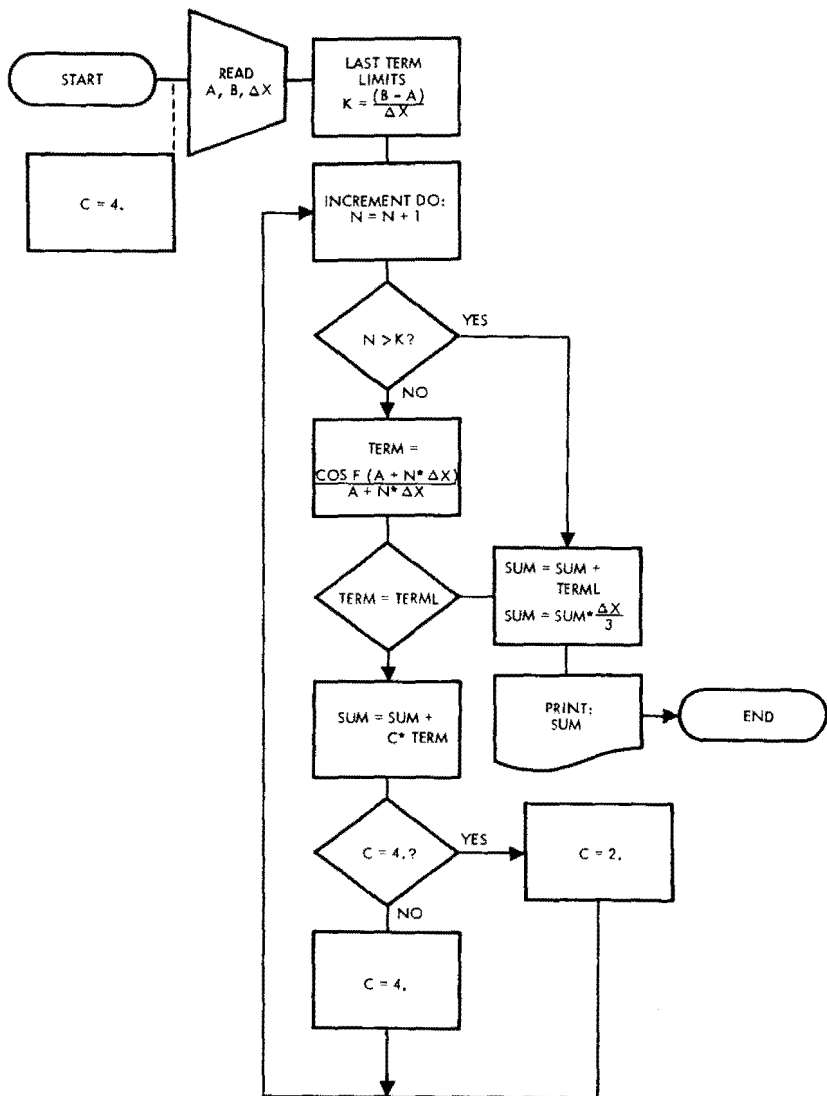
for the following possible values:

<u>Variable</u>	<u>Range of Values</u>
a	-6.99 to +6.99
b	-6.99 to +6.99
Δx	-.25 to +.25

Simpson's rule for approximating a definite integral is:

$$\int_a^b f(x)dx = \frac{\Delta x}{3} (f(a)+4f(a+\Delta x)+2f(a+2\Delta x)+4f(a+3\Delta x)+\dots +f(b))$$

The last term is reached when $(a+k\Delta x)=b$, and when neither a 2 nor a 4 appears in front of the first or last term.



SAMPLE PROGRAM FLOWCHART

PROGRAMMER		DATE	PROGRAM
C	Line	10	20
FTN	B, L, A		
	PROGRAM SMPSN		
	READ(1,10) A,B,DELTX		
10	FORMAT(2E8.2,E7.2)		
	TERML=cos(B)/B		
	SUM=cos(A)/A		
	K=(B-A)/DELTX		
	C=4.		
	I=K+1		
	DO 60 N=1,I		
	FN=N		
	IF(N-K)20,20,70		
20	TERM=cos(A+FN*DELTX)/(A+FN*DELTX)		
	IF(TERM-TERML)30,70,30		
30	SUM=SUM+C*TERM		
	IF(C-4.)50,40,50		
40	C=2.		
	GO TO 60		
50	C=4.		
60	CONTINUE		
70	SUM=SUM+TERML		
80	SUM=(SUM*DELTX)/3.		
	WRITE(2,90) SUM		
90	FORMAT("SUM=",E8.2)		
	STOP		
	END		
	ENDS		

0 - ZERO

0 - ALPHA 0

1 - ONE

1 - ALPHA 1

LINE TERMINATED BY RETURN LINE FEED (LF)

2 - TWO

2 - ALPHA 2

LINE IS DELETED BY RETURN BEFORE E-11

SOURCE PROGRAM LISTING

```
FTN,B,L,A
PROGRAM SMPSN
READ(1,10) A,B,DELTX
10 FORMAT(2E8.2,E7.2)
TERML=cos(B)/B
SUM=cos(A)/A
K=(B-A)/DELTX
C=4.
I=K+1
DO 60 N=1,I
FN=N
IF(N-K)20,20,70
20 TERM=cos(A+FN*DELTX)/(A+FN*DELTX)
IF(TERM-TERML)30,70,30
30 SUM=SUM+C*TERM
IF(C-4.)50,40,50
40 C=2.
GO TO 60
50 C=4.
60 CONTINUE
70 SUM=SUM+TERML
80 SUM=(SUM*DELTX)/3.
WRITE(2,90) SUM
90 FORMAT("SUM=",E8.2)
STOP
END
END$
```

OBJECT PROGRAM LISTING

Assembly Level Language

PAGE 0001 SMPN

```

00000 000000 BSS 000000
00000 000000 OCT 000000
00001 062314R LDA 000314
00002 006404 OCT 006404
00003 016001X JSB .DIO.
00004 100277R DEF 000277,1
00005 100300R DEF 000300,1
00006 016002X JSB .10R.
00007 016003X JSB .DST
00010 000246R DEF 000246
00011 016002X JSB .10R.
00012 016003X JSB .DST
00013 000250R DEF 000250
00014 016002X JSB .10R.
00015 016003X JSB .DST
00016 000252R DEF 000252
00017 126301R JMP 000301,1
00020 024040 OCT 024040
00021 031105 OCT 031105
00022 034056 OCT 034056
00023 031054 OCT 031054
00024 042467 OCT 042467
00025 027062 OCT 027062
00026 024400 OCT 024400
00027 016004X JSB .DLD
00030 000250R DEF 000250
00031 016005X JSB COS
00032 016006X JSB .FDV
00033 000250R DEF 000250
00034 016003X JSB .DST
00035 000254R DEF 000254
00036 016004X JSB .DLD
00037 000246R DEF 000246
00040 016005X JSB COS
00041 016006X JSB .FDV
00042 000246R DEF 000246
00043 016003X JSB .DST
00044 000256R DEF 000256
00045 016004X JSB .DLD
00046 000250R DEF 000250
00047 016007X JSB .FSB
00050 000246R DEF 000246
00051 016006X JSB .FDV
00052 000252R DEF 000252

```

```

00053 016010X JSB IFIX
00054 072260R STA 000260
00055 016004X JSB .DLD
00056 000315R DEF 000315
00057 016003X JSB .DST
00060 000261R DEF 000261
00061 062260R LDA 000260
00062 042314R ADA 000314
00063 072263R STA 000263
00064 062314R LDA 000314
00065 072264R STA 000264
00066 062264R LDA 000264
00067 016011X JSB FLOAT
00070 016003X JSB .DST
00071 000265R DEF 000265
00072 062264R LDA 000264
00073 003004 OCT 003004
00074 042260R ADA 000260
00075 003004 OCT 003004
00076 002020 OCT 002020
00077 126302R JMP 000302,I
00100 002002 OCT 002002
00101 126303R JMP 000303,I
00102 126302R JMP 000302,I
00103 016004X JSB .DLD
00104 000265R DEF 000265
00105 016012X JSB .FMP
00106 000252R DEF 000252
00107 016013X JSB .FAD
00110 000246R DEF 000246
00111 016003X JSB .DST
00112 000271R DEF 000271
00113 016005X JSB COS
00114 016003X JSB .DST
00115 000273R DEF 000273
00116 016004X JSB .DLD
00117 000265R DEF 000265
00120 016012X JSB .FMP
00121 000252R DEF 000252
00122 016013X JSB .FAD
00123 000246R DEF 000246
00124 016003X JSB .DST
00125 000275R DEF 000275
00126 016004X JSB .DLD
00127 000273R DEF 000273
00130 016006X JSB .FDV
00131 000275R DEF 000275
00132 016003X JSB .DST
00133 000267R DEF 000267
00134 016007X JSB .FSB
00135 000254R DEF 000254
00136 002020 OCT 002020
00137 126304R JMP 000304,I
00140 002002 OCT 002002

```

```

00141 126304R JMP 000304,I
00142 126303R JMP 000303,I
00143 016004X JSB .DLD
00144 000261R DEF 000261
00145 016012X JSB .FMP
00146 000267R DEF 000267
00147 016013X JSB .FAD
00150 000256R DEF 000256
00151 016003X JSB .DST
00152 000256R DEF 000256
00153 016004X JSB .DLD
00154 000261R DEF 000261
00155 016007X JSB .FSB
00156 000315R DEF 000315
00157 002020 OCT 002020
00160 126305R JMP 000305,I
00161 002002 OCT 002002
00162 126305R JMP 000305,I
00163 126306R JMP 000306,I
00164 016004X JSB .DLD
00165 000317R DEF 000317
00166 016003X JSB .DST
00167 000261R DEF 000261
00170 126307R JMP 000307,I
00171 016004X JSB .DLD
00172 000315R DEF 000315
00173 016003X JSB .DST
00174 000261R DEF 000261
00175 062264R LDA 000264
00176 002004 OCT 002004
00177 072264R STA 000264
00200 003004 OCT 003004
00201 042263R ADA 000263
00202 002021 OCT 002021
00203 026066R JMP 000066
00204 016004X JSB .DLD
00205 000256R DEF 000256
00206 016013X JSB .FAD
00207 000254R DEF 000254
00210 016003X JSB .DST
00211 000256R DEF 000256
00212 016004X JSB .DLD
00213 000256R DEF 000256
00214 016012X JSB .FMP
00215 000252R DEF 000252
00216 016006X JSB .FDV
00217 000321R DEF 000321
00220 016003X JSB .DST
00221 000256R DEF 000256
00222 062323R LDA 000323
00223 006400 OCT 006400
00224 016001X JSB .DIO.
00225 100311R DEF 000311,I
00226 100312R DEF 000312,I

```

00227	016004X	JSB .DLD
00230	000256R	DEF 000256
00231	016002X	JSB .IOR.
00232	016014X	JSB .DTA.
00233	126313R	JMP 000313, I
00234	024040	OCT 024040
00235	021123	OCT 021123
00236	052515	OCT 052515
00237	036442	OCT 036442
00240	026105	OCT 026105
00241	034056	OCT 034056
00242	031051	OCT 031051
00243	002400	OCT 002400
00244	016015X	JSB .STOP
00245	016015X	JSB .STOP
00246	000000	BSS 000030
00276	000000	OCT 000000
00277	000020R	DEF 000020
00300	000017R	DEF 000017
00301	000027R	DEF 000027
00302	000103R	DEF 000103
00303	000204R	DEF 000204
00304	000143R	DEF 000143
00305	000171R	DEF 000171
00306	000164R	DEF 000164
00307	000175R	DEF 000175
00310	000212R	DEF 000212
00311	000234R	DEF 000234
00312	000233R	DEF 000233
00313	000243R	DEF 000243
00314	000001	OCT 000001
00315	040000	OCT 040000
00316	000006	OCT 000006
00317	040000	OCT 040000
00320	000004	OCT 000004
00321	060000	OCT 060000
00322	000004	OCT 000004
00323	000002	OCT 000002
		TRA SMPSN

*** END

OBJECT PROGRAM LISTING

Symbol Table

PAGE 0005 SMPSN

SYMBOL TABLE	
SMPSN	R 000000
A	R 000246
B	R 000250
DELTX	R 000252
00010	R 000020
10000	R 000017
10001	R 000027
TERML	R 000254
SUM	R 000256
K	R 000260
C	R 000261
I	R 000263
N	R 000264
FN	R 000265
00020	R 000103
00070	R 000204
TERM	R 000267
00030	R 000143
00050	R 000171
00040	R 000164
00060	R 000175
00080	R 000212
00090	R 000234
10002	R 000233
10003	R 000243

BASIC CONTROL SYSTEM

Relocating Loader Memory Allocation

SMPSN

02000 02323

LOAD

FRMTR

**02324 04234
00240 00730**

COS

04235 04244

SIN

04245 04346

CHEBY

04347 04437

• FCM

04440 04447

• DLC

04450 04461

E-10 FORTRAN

FADSB

04462 04617

FDV

04620 04723

FMP

04724 05007

MPY

05010 05120

.IENT

05121 05155

FLOAT

05156 05162

.PACK

05163 05267

DIV

05270 05362

DLDST

05363 05420

IFIX

05421 05455

•STOP

05456 05476

•ERRR

05477 05517

PWR2

05520 05543

•FLUN

05544 05556

***LST**

.IOC. 15434
.SQT. 15405
.MEM. 15400
.BUFR 15602
SMPSN 02000
.DIO. 03635
.IOR. 03505
.DST 05373
.DLD 05363
COS 04235
.FDV 04620
.FSB 04465
IFIX 05421
FLOAT 05156
.FMP 04724
.FAD 04462
.DTA. 03733
.STOP 05456
.BIO. 03710
.IOI. 03532
.IAR. 03571
.RAR. 03545
.FLUN 05544
.PACK 05163
.MPY 05010
SIN 04245
..FCM 04440
.ERRR 05477
.CHEB 04347
.IENT 05121
.PWR2 05520
..DLC 04450
.DIV 05270

***LINKS**

01723 01777

***RUN**

OBJECT PROGRAM **Input and Output Data**

```

      1.23      4.72      .25
SUM=-.63E+00
STOP
      1.23      2.01      .10
SUM=-.12E-01
STOP
      0.34      1.01      .02
SUM= .88E+00
STOP
      0.00      1.00      .01
SUM= .57E+36
STOP
      1.00      1.25      .05
SUM= .92E-01
STOP

```

INDEX

- Arithmetic expressions 2-6, 3-1
- Arithmetic operators 3-1
- Arguments
 - Actual 6-1, 6-2, 6-4, 6-5, 6-7
 - Dummy 6-1, 6-2, 6-3, 6-5, 6-9
 - Redefinition 6-3, 6-5
- Array 2-3, 2-4, 2-5, 2-6, 4-1, 4-6,
6-2, 7-2
- ASA Basic FORTRAN v
- ASCII
 - Input data 1-1, 7-1, 8-1
 - FORMAT specifications 7-10,
7-12
 - Output data 7-1
- Assembler source program C-1
- Assembly level listing 9-1, 9-2
- Assignment statements 3-4

- BACKSPACE statement 8-3, 8-4
- Basic Binary Loader 9-4
- Basic Control System v, 1-2, 8-1, 9-7,
9-8, C-1
- Basic External Functions 3-5, 6-1,
6-9, 6-11
 - IAND 3-5
 - IOR 3-5
 - NOT 3-5

- CALL statement 6-4, 6-12, C-1
- Calling program 6-1
- Character set
 - FORTRAN 1-1
 - HP 2116A A-1
- Coding Form 1-3, 1-4
- Commercial at (@) 1-1, 7-4, 7-11,
7-16, 7-17
- COMMON statement 2-5, 4-1, 4-2,
4-3, 4-6, 4-7,
6-1, 6-2

Comments 1-2, 7-18
 Compiler v, 9-1
 Constant
 Integer 2-2, 5-3, 7-1, 7-5, 8-1
 Octal 2-2
 Real 2-3
 Continuation lines 1-2
 CONTINUE statement 5-9, 8-3, 8-4
 Control statement, compiler 1-2, 9-1
 Crossbar scanner D-1, D-6

Data item delimiters 7-16
 Data Source Interface Card D-1
 Diagnostics

 Compilation 9-8, 9-11
 Library 9-19
 Object program 9-17
 Source program 9-12

Digital voltmeter D-1
 DO-Implied list 7-2, 7-3
 DO Loop 5-4, 5-7, 5-8, 5-9, 6-4, 7-2,
 7-3
 DO Nest 5-6, 5-7, 7-3
 DO Statement 5-3
 Dollar sign (\$) 7-5, 7-9
 DIMENSION statement 2-5, 4-1, 4-2,
 6-2
 DVM Programmer Card D-1, D-3

END, END\$ statements 1-3, 5-10,
 6-3, 6-4, 6-5
 End-of-statement mark 1-2, 1-3,
 (CR) (LF) 7-18, 9-1
 ENDFILE statement 8-3, 8-4
 ENTR C-4
 EQUIVALENCE statement 4-1, 4-5,
 4-6, 4-7,
 4-8, 4-9,
 6-2
 Evaluation of expressions 3-2, 6-5,
 6-9
 Exponent 2-1, 7-5, 7-6
 Expressions 2-6, 3-1, 3-2, 3-3, 3-4,
 5-2, 6-2, 6-7, 6-9

Fixed Decimal 2-1
 Floating decimal (point) 2-1, 7-17
 FORMAT statement 1-1, 5-1, 7-1, 7-4,
 8-1, 8-2, 9-1
 Specifications 7-4, 7-8, 7-12
 Aw 7-10
 Ew.d input 7-6
 Ew.d output 7-5
 Fw.d input 7-9
 Fw.d output 7-8
 Iw 7-9
 Kw 7-11
 nX 7-12
 nH 7-12
 r@w 7-11
 r"... " 7-13
 Fraction 2-1, 7-5, 7-6, 7-9
 FTN
 Control statement 9-1
 Program name 6-2
 Free Field input v, 1-1, 7-16, 8-1
 Function
 Basic External 3-5, 6-1, 6-9,
 6-11
 Reference 6-1, 6-7, 6-9, 6-12,
 C-1
 Statement 6-1, 6-9
 Subprogram 6-1, 6-5, 6-9
 FUNCTION statement 6-3, 6-5

 GO TO statements
 Computed 5-1
 Unconditional 5-1

 Hierarchy of operations 3-2

 IF statements 5-2, 5-3, 5-8
 Three-branch 5-2
 Two-branch 5-3

 Input/Output
 List 7-1, 7-3
 Statements 8-1, 8-3

Integer

Array 2-5, 4-6
Constant 2-2, 5-1, 5-3, 7-1,
7-5, 8-1
Quantity 2-1, 7-4, 7-6, 7-9,
7-10
Statement 3-4
Variable 2-3, 4-6, 5-1, 8-1

Labels 1-2, 5-1, 5-2
Library 6-1, 6-11, 9-2, 9-8, 9-20
Line 1-2
List 7-1, 7-2, 7-16, 7-18, 8-1, 8-2, 9-2

Main program 6-1, 6-2
Masking operations v, 3-5
Memory Allocation Listing 9-7

Object listing 9-1
Object program v, 6-11, 9-1, 9-7, 9-15,
9-17, C-1

Octal

Constants v, 2-2
Input data 7-17

Operating instructions

Magnetic tape 9-10
Paper tape 9-4

Parameters

Control statement 9-1
Indexing (DO) 5-3, 7-2
Initial (DO) 5-3, 7-3
Subprogram 6-1, 6-5
Terminal (DO) 5-3, 7-3

Parentheses 3-2, 7-4, 7-16
Pass v, 9-1, 9-9, 9-10
PAUSE statement 5-3, 5-9
PROGRAM statement 6-2

Quotation marks 7-13, 7-18

READ statement 7-1
 Formatted 8-2
 Free Field 8-2
 Unformatted 8-3
 Real
 Array 2-5, 4-6
 Expression 3-3
 Quantity 2-1, 2-3, 7-1, 7-8
 Statement 3-4
 Variable 2-3, 4-6
 Record 7-4, 7-12, 8-1
 Relocatable binary 1-2, 9-1, 9-2, 9-4
 Relocation indicator 9-3
 Repeat specification 7-15, 7-16
 RETURN statement 5-3, 6-3, 6-4, 6-5,
 6-7, 6-12
 REWIND statement 8-3

 Slash (/) 7-4, 7-17
 Source listing 1-2, 9-1
 Source program v, 5-3, 9-1, 9-2, C-1
 Spaces (blanks) 1-1, 2-3, 7-12, 9-1
 Standard units 8-1
 Input 9-4
 List Output 9-2, 9-4
 Program library 9-15
 Teleprinter output 5-9, 5-10
 Statement labels 1-2, 2-7, 5-1, 5-2
 STOP statement 5-3, 5-9, 6-12
 Statement function 6-1, 6-9
 Subprograms
 Function 6-1, 6-5, 6-9, 6-12, C-1
 Subroutine 6-1, 6-3, 6-12, C-1
 Subroutine
 Call 6-2, 6-4
 Subprogram 6-2, 6-3, 6-12, C-1
 SUBROUTINE statement 6-3
 Subscripts 2-4, 2-5, 7-1, 7-2
 Symbol table 9-2, 9-3
 System Input/Output (SIO) 9-5

Type

Arguments 6-1
Array 2-5, 4-1
Expression 2-6, 3-3
Statement 3-4
Variable 2-3

Unlimited groups 7-16

Unit-reference number 8-1

Variables 3-4, 6-4, 6-9

Control 7-2, 7-3

Integer 2-3, 4-6, 5-1, 5-3

Real 2-3, 4-6

Simple 2-3, 7-1

Subscripted 2-4

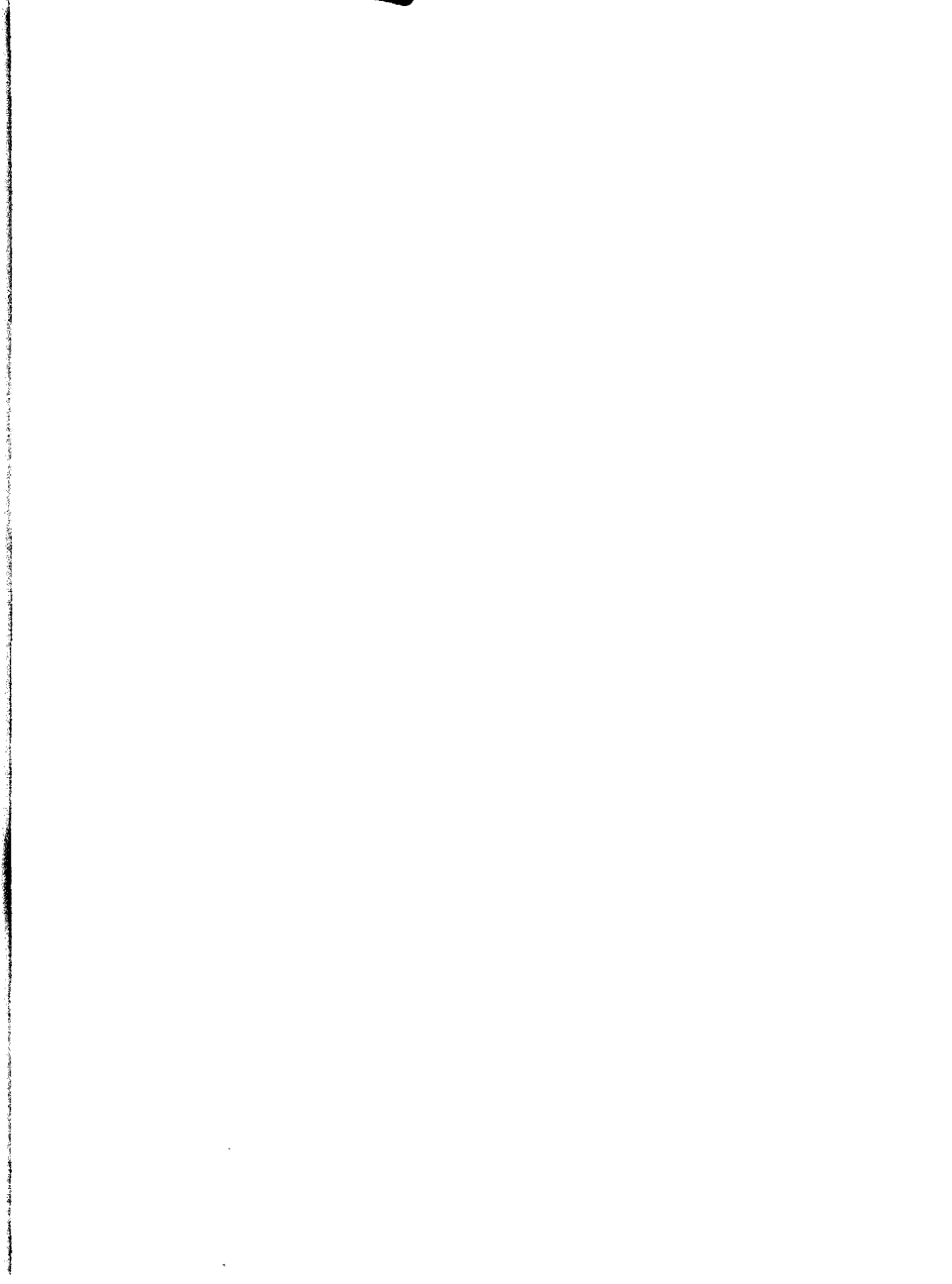
WRITE statement 7-1

Formatted 8-2

Unformatted 8-3



Program Library Reference Manual



CONTENTS

INTRODUCTION	iii/iv
------------------------	--------

CHAPTER 1

MATH ROUTINES	1-1
ABS	1-1
ALOG	1-2
ATAN	1-3
COS	1-4
EXP	1-5
FLOAT	1-6
IABS	1-7
IFIX	1-8
ISIGN	1-9
SIGN	1-10
SIN	1-11
SQRT	1-12
TAN	1-13
TANH	1-14

CHAPTER 2

ARITHMETIC AND EXPONENTIAL ROUTINES . . .	2-1
.DIV	2-1
.DLD	2-2
.DST	2-3
.FAD	2-4
.FSB	2-5
.FDV	2-6
.FMP	2-7
.ITOI	2-8
.MPY	2-10
.RTOI	2-11
.RTOR	2-12

CHAPTER 3

INPUT/OUTPUT SERVICES ROUTINES	3-1
CLRIO	3-1
ENDIO	3-2
FRMTR	3-3
LEADR	3-6
PTAPE	3-7

CHAPTER 4

SERVICE ROUTINES	4-1
CHEBY	4-1
.FCM	4-2
.ERRR	4-3
.FLUN	4-4
.IENT	4-5
ISSW	4-6
MANT	4-7
.PACK	4-8
PWR2	4-9

CHAPTER 5

MISCELLANEOUS ROUTINES	5-1
.DLC	5-1
ENTIE	5-2
.ENTR	5-3
.GOTO	5-4
IAND	5-5
INDEX	5-6
IOR	5-7
.MAP	5-8
MEMRY	5-9
OVF	5-10
.PAUS	5-11
.PRAM	5-12
.RND	5-13
.STOP	5-14
.SWCH	5-15
.TAPE	5-16

APPENDIX A

SUMMARY OF ERROR CONDITIONS AND CODES . . .	A-1
---	-----

APPENDIX B

ALPHABETICAL LISTING OF SUBROUTINES . . .	B-1
---	-----

INTRODUCTION

This manual contains descriptions of the HP Computer Program library routines. These routines are grouped into five general categories: Math Routines, Arithmetic and Exponential Routines, Input/Output Service Routines, and Miscellaneous Routines.

Many of the routines including Input/Output Service Routines may be called from a FORTRAN or ALGOL program by specification of a function of the general form:

$$\text{function } (p_1, p_2, \dots, p_n)$$

The Arithmetic and Exponential Routines are called from a program as a result of arithmetic expressions in the language. For example, $I**K$ would generate a call to the .ITOI routine.

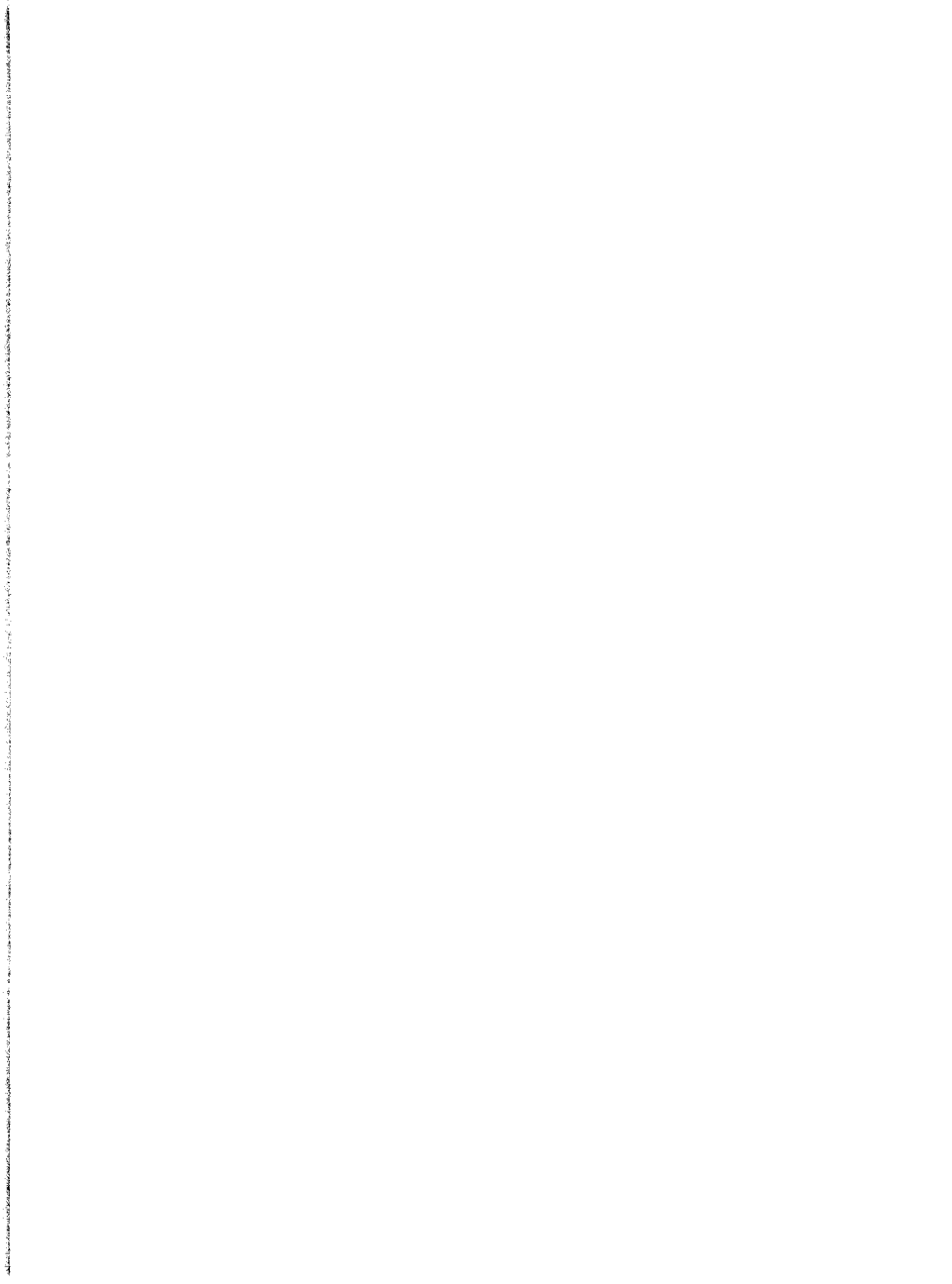
The Input/Output Service Routines are those which are called in FORTRAN and ALGOL programs. The Service Routines are those which are called by the other classes of Program library routines.

The Miscellaneous Routine performs various functions.

All classes of Program library routines may be called from an Assembly-language program. The Assembler calling sequence is listed with the description; routine names must be declared as EXTERNAL points in the program.

When a FORTRAN, ALGOL, or Assembly-language object program which references Program library routines is loaded for execution, Program library must also be loaded. The BCS Relocating Loader will load only those routines called by the user's object program.

The execution times listed in this manual are approximate. The i , j , x , y , and z notations indicate variable names or array element names to be provided by the user in the function or calling sequence. The variables i and j indicate integer, or fixed point, values; x and y indicate real, floating point, values. z is used to indicate that the value may be integer or real.



ABS

PURPOSE: To calculate the absolute value of x , where x has a floating point value.

CALLING SEQUENCE: DLD x
JSB ABS

FORTTRAN FUNCTION: ABS(x)

NORMAL RETURN: Floating point result is left in the A and B registers.

STORAGE: 5 words

EXECUTION TIME: For $x \geq 0$, .006 ms
For $x < 0$, 100 μ s

ACCURACY: 23 bits

ALOG

PURPOSE:

To calculate the natural logarithm of x, where x has a floating point value.

CALLING SEQUENCE:

DLD x

JSB ALOG

FORTTRAN FUNCTION:

ALOG(x)

NORMAL RETURN:

Floating point result is left in the A and B registers. If error condition (below) occurs, the overflow bit is set.

STORAGE:

70 words (106₈)

EXECUTION TIME:

Minimum = 6.12 ms

Average = 7.7 ms

Maximum = 10.65 ms

ACCURACY:

22 bits

ERROR CONDITIONS:

Condition

Error Code

$x \leq 0$

ø2 UN

ALGORITHM:

Let f = mantissa (x)

i = characteristic (x)

(that is, $x = 2^i \times f$)

Then answer = $(i + \log_2 f) \cdot (\log_e 2)$

$$= \log_e 2 \left(i + z \left[c_1 + \frac{c_2}{c_3 - z^2} \right] - 1/2 \right)$$

where

$$z = \frac{f - \sqrt{2}/2}{f + \sqrt{2}/2}$$

and

$$c_1 = 1.2920070987$$

$$c_2 = 2.6398577035$$

$$c_3 = 1.6567626301$$

PURPOSE: To calculate the Arctangent of x , where x has a floating point value.

CALLING SEQUENCE: DLD x
JSB ATAN

FORTTRAN FUNCTION: ATAN(x)

NORMAL RETURN: Result is in radians, in the range $-\pi/2$ to $\pi/2$. Floating point result is left in A and B registers.

STORAGE: 87 words (127_8)

EXECUTION TIME: Minimum = 15.82 ms
Average = 20.9 ms
Maximum = 28.8 ms

ACCURACY: 21 bits

ALGORITHM: if $\text{abs}(X) > 1$ then $U = 1/X$ else $U = X$
 $Y = U * \text{Cheby}(2 * U * U - 1)$
if $\text{abs}(X) \leq 1$ then answer = Y
else if $X > 0$ then answer = $\pi/2 - Y$
else answer = $-\pi/2 - Y$

COS

PURPOSE: To calculate the cosine of x , where x has a floating point value expressed in radians.

CALLING SEQUENCE: DLD x
JSB COS

FORTRAN FUNCTION: COS(x)

NORMAL RETURN: Floating point result is left in A and B registers. If error condition (below) occurs, the overflow bit is set.

STORAGE: 8 words (10_8)

EXECUTION TIME: Minimum = 10.26 ms
Average = 13.8 ms
Maximum = 20.25 ms

ACCURACY: See SIN.

ERROR CONDITIONS:	<u>Condition</u>	<u>Error Code</u>
	$ABS(x) > 2^{14}$	05 OR

ALGORITHM: $COS(x) = -SIN(x - \pi/2)$

PURPOSE: To calculate e^x , where x has a floating point value.

CALLING SEQUENCE: DLD x
JSB EXP

FORTTRAN FUNCTION: EXP(x)

NORMAL RETURN: Floating point result is left in the A and B registers. If error condition (below) occurs, the overflow bit is set.

STORAGE: 87 words (127₈)

EXECUTION TIME: Minimum = 6.12 ms
Average = 7.7 ms
Maximum = 10.65 ms

ACCURACY: $e^{x+y} = e^x * e^y$

Hence, an absolute error of y in the representation of x introduces an error of $(e^{x+y} - e^x)$ into the value of EXP(x).

Thus for values of x with $|x| < 1$, 23 bits of accuracy can be expected, while for values of x with $|x|$ near 100, only 15 bits of accuracy can be expected.

ERROR CONDITIONS:	Condition	Error Code
	$x * \log_2 e \geq 124$	07 OF

ALGORITHM: Let $i = \text{ENTIER}(x)$, and $f = x * \log_2 e - 1$ (See .IENT)

$$\text{Answer} = 2^i * \left[1 + \frac{2f}{c_4 + c_3 f^2 - f - c_2 / (f^2 + c_1)} \right]$$

where

$$\begin{aligned} c_1 &= 87.417497202 \\ c_2 &= 617.9722695 \\ c_3 &= 0.03465735903 \\ c_4 &= 9.9545957821 \end{aligned}$$

FLOAT

PURPOSE:	To convert the integer <i>i</i> to floating point format.
CALLING SEQUENCE:	LDA <i>i</i> JSB FLOAT
FORTTRAN FUNCTION:	FLOAT(<i>i</i>)
NORMAL RETURN:	Floating point result is left in the A and B registers.
EXECUTION TIME:	Minimum = .1 ms Average = .2 ms Maximum = .3 ms
STORAGE:	10 words (12 ₈)

PURPOSE:	To calculate absolute value of i , where i has an integer value.
CALLING SEQUENCE:	LDA i JSB IABS
FORTRAN FUNCTION:	IABS (i)
NORMAL RETURN:	Integer result is left in A register.
EXECUTION TIME:	.0096 ms
STORAGE:	7 words
ACCURACY:	23 bits
ERROR CONDITIONS:	None

IFIX

PURPOSE: To convert the floating point number x to integer format.

CALLING SEQUENCE: DLD x
JSB IFIX

FORTTRAN FUNCTION: IFIX(x)

NORMAL RETURN: Integer result is left in A register. Any fractional portion is truncated. If the integer portion is greater than or equal to 2^{15} , the result is set to 32767.

STORAGE: 29 words (35_8)

PURPOSE:	To calculate sign of z times $ i $ where z has an integer or floating point value and i has an integer value.
CALLING SEQUENCE:	JSB ISIGN DEF i DEF z
FORTRAN FUNCTION:	ISIGN (i, z)
NORMAL RETURN:	Integer result is left in A register.
STORAGE:	26 words (32_8)
EXECUTION TIME:	.05 ms
ACCURACY:	23 bits
ALGORITHM:	Same as SIGN.

SIGN

PURPOSE:

To calculate sign of y times $|x|$, where x has a floating point and z has a floating point or integer value. In the case where $z=0$, the result of the SIGN function is 0, regardless of the value of $|x|$.

CALLING SEQUENCE:

JSB SIGN
DEF x
DEF z

FORTRAN FUNCTION:

SIGN (x, z)

NORMAL RETURN:

Floating point result is left in A and B registers.

STORAGE:

30 words (36_8)

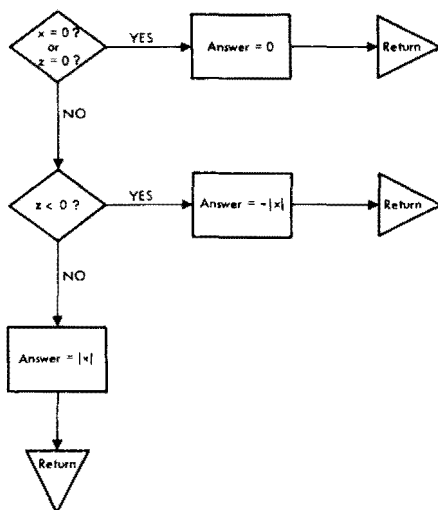
EXECUTION TIME:

Minimum = .3 ms
Average = .5 ms
Maximum = .9 ms

ACCURACY:

23 bits

ALGORITHM:



PURPOSE: To calculate sine of x , where x has a floating point value expressed in radians.

CALLING SEQUENCE: DLD x
JSB SIN

FORTTRAN FUNCTION: SIN(x)

NORMAL RETURN: Floating point result is left in the A and B registers. If error condition (below) occurs, the overflow bit is set.

STORAGE: 66 words (102_8)

EXECUTION TIME: Minimum = 10.26 ms
Average = 13.8 ms
Maximum = 20.25 ms

ACCURACY: For small y , we have $\sin(n\pi + y) = \sin(y) \approx y$
That is, the accuracy of $\sin(x)$ for x near a non-zero integer multiple of π is limited by the accuracy of the subtraction $n\pi - x$.
For example, $\sin(3.14)$ would have approximately 4 decimal digits of accuracy (the possible 7 minus the 3-digit closeness to π).

ERROR CONDITIONS:	<u>Condition</u>	<u>Error Code</u>
	$\text{ABS}(x) > 2^{14}$	05 OR

ALGORITHM: $X = X * 2 / \pi$
 $X = X - 4 * \text{ENTIER}((X+1)/4)$ (See .IENT)
 if $X > 1$ then $X = 2 - X$
 answer = $X * \text{Cheby}(2 * X * X - 1)$

SQRT

PURPOSE: To calculate the square root of x , where x has a floating point value.

CALLING SEQUENCE: DLD x
JSB SQRT

FORTTRAN FUNCTION: SQRT(x)

NORMAL RETURN: Floating point result is left in A and B registers. If error condition (below) occurs, the overflow bit is set.

STORAGE: 77 words (115_8)

EXECUTION TIME: Minimum = 3.94 ms
Average = 4.8 ms
Maximum = 6.45 ms

ACCURACY: 22 bits

ERROR CONDITIONS:

<u>Condition</u>	<u>Error Code</u>
$x < 0$	03 UN

ALGORITHM: Choose f such that $x = 2^{2b}(f)$, $.25 \leq f < 1$
Then $\sqrt{x} = 2^b \sqrt{f}$
 \sqrt{f} is approximated by $p_1 = c_1 f + c_2$, where for
 $.25 \leq f < .5$, $c_1 = .875$, $c_2 = .27863$
and for $.5 \leq f < 1$, $c_1 = .578125$, $c_2 = .421875$
This approximation is improved by two Newton iterations:
$$p_2 = (p_1 + f/p_1)/2$$
$$p_3 = (p_2 + f/p_2)/2$$
$$p_3 \text{ is the final result}$$

PURPOSE: To calculate tangent of x , where x has a floating point value expressed in radians.

CALLING SEQUENCE: DLD x
JSB TAN

FORTTRAN FUNCTION: TAN(x)

NORMAL RETURN: Floating point result is left in the A and B registers. If error condition (below) occurs, the overflow bit is set.

STORAGE: 87 words (127_8)

EXECUTION TIME: Minimum = 14.28 msec.
Average = 19.4 msec.
Maximum = 28.8 msec.

ACCURACY: Approximately as accurate as $\text{SIN}(x)/\text{COS}(x)$.

ERROR CONDITIONS:	<u>Condition</u>	<u>Error Code</u>
	$x > 2^{14}$	09 OR
	$\tan(x) > 2^{128}$	(floating point overflow)

ALGORITHM: $X = 4 * X / \pi$
 $X = X - 4 * \text{ENTIER}((X+1)/4)$ (See .IENT)
 if $X > 1$ then $Y = 2 - X$ else $Y = X$.
 $Y = Y * \text{CHEBY}(2 * Y * Y - 1)$
 if $X > 1$ then answer = $1/Y$ else answer = Y .

TANH

- PURPOSE:** To calculate hyperbolic tangent of x , where x has a floating point value.
- CALLING SEQUENCE:** DLD x
JSB TANH
- FORTTRAN FUNCTION:** TANH(x)
- NORMAL RETURN:** Floating point result is left in the A and B registers.
- STORAGE:** 99 words (143₈)
- EXECUTION TIME:** Depends on the range in which x lies. Always less than twice as long as EXP.
- ACCURACY:** At least 20 bits correct.
- ALGORITHM:**
1. $x \geq 0$
 - a. $x \geq 16$
$$\text{TANH}(x) = 1$$
 - b. $.125 \leq x < 16$
$$\text{TANH}(x) = (\text{EXP}(2 \cdot x) - 1) / (\text{EXP}(2 \cdot x) + 1)$$
 - c. $.00005 \leq x < .125$
$$\text{TANH}(x) = \left\{ c_1 + f^2 \left[c_2 + c_3 (c_4 + f^2)^{-1} \right] \right\}^{-1}$$

where:

$$f = 4 \cdot x \cdot \log_2 e$$
$$c_1 = 5.7707801636$$
$$c_2 = .01732867951$$
$$c_3 = 14.1384114018$$
$$c_4 = 349.6699888$$
 - d. $x < .00005$
$$\text{TANH}(x) = x$$
 2. $x < 0$
$$\text{TANH}(x) = -\text{TANH}(-x)$$

ARITHMETIC AND EXPONENTIAL ROUTINES 2

.DIV

PURPOSE: To divide the two-word integer quantity *i* by the integer quantity *j*.

CALLING SEQUENCE: JSB .DIV
DEF *j*

ASSEMBLY LANGUAGE: DIV *j* (*i* is a two-word quantity in the B and A registers)

NORMAL RETURN: Result is left in the A and B registers; the quotient is left in A, the remainder in B.

STORAGE: 49 words (61₈)

EXECUTION TIME: Minimum = .27 ms
Average = .3 ms
Maximum = .31 ms

ERROR CONDITIONS: If the quotient is outside the range [-32768, 32767], the overflow flag is set and the accumulator is set to 32767.

.DLD

PURPOSE:	To load x, a two-word quantity, into the A and B registers.
CALLING SEQUENCE:	JSB .DLD DEF x (x = address of the first word of the two-word quantity)
ASSEMBLY LANGUAGE:	DLD x (x = address of the first word of the two-word quantity)
NORMAL RETURN:	The quantity in locations x and x+1 will be loaded into the A and B registers.
STORAGE:	30 words (36 ₈)
EXECUTION TIME:	.06 ms

PURPOSE: To store a two word quantity from the A and B registers to two consecutive locations in memory.

CALLING SEQUENCE: JSB .DST
DEF x

ASSEMBLY LANGUAGE: DST x

NORMAL RETURN: The quantity in the A and B registers will be stored in locations x and x+1 .

STORAGE: 30 words (36_8)

EXECUTION TIME: .06 ms

.FAD

PURPOSE: To add two floating point numbers, x and y.

CALLING SEQUENCE: JSB .FAD

DEF y (x is assumed to be in A and B)

ASSEMBLY LANGUAGE: FAD y (x is assumed to be in A and B)

NORMAL RETURN: The floating point sum is left in the A and B registers.

STORAGE: 94 words (136₈)

EXECUTION TIME: Minimum = .3 ms

Average = .5 ms

Maximum = .9 ms

ERROR CONDITIONS:

If the result is outside the range of representable floating point numbers: $[-2^{127}, 2^{127}(1-2^{-23})]$ the overflow flag is set and the result $2^{128}(1-2^{-23})$ is returned. If an underflow occurs, (result within the range $(-2^{-129}(1+2^{-22}), 2^{-129})$) the overflow flag is set and the result 0 is returned.

PURPOSE:	To subtract the floating point number y from the floating point number x.
CALLING SEQUENCE:	JSB .FSB DEF y (x is assumed in A and B)
ASSEMBLY LANGUAGE:	FSB y (x is assumed in A and B)
NORMAL RETURN:	The floating point difference (x-y) is placed in the A and B registers.
STORAGE:	94 words (136 ₈)
EXECUTION TIME:	Minimum = .3 ms Average = .5 ms Maximum = .9 ms
ERROR CONDITIONS:	See FAD, page 2-4.

.FDV

PURPOSE:	To divide the floating point quantity x by the floating point quantity y.
CALLING SEQUENCE:	JSB .FDV DEF y (x is assumed in A and B)
ASSEMBLY LANGUAGE:	FDV y (x is assumed in A and B)
NORMAL RETURN:	The floating point quotient is left in the A and B registers.
STORAGE:	70 words (104 _g)
EXECUTION TIME:	Minimum = 1.2 ms Average = 1.3 ms Maximum = 1.5 ms
ERROR CONDITIONS:	See FAD, page 2-4.

PURPOSE: To multiply the two floating point numbers x and y.

CALLING SEQUENCE: JSB .FMP
DEF x (y is assumed in A and B)

ASSEMBLY LANGUAGE: FMP x (y is assumed in A and B)

NORMAL RETURN: The floating point product is returned in the A and B registers.

STORAGE: 51 words (63_8)

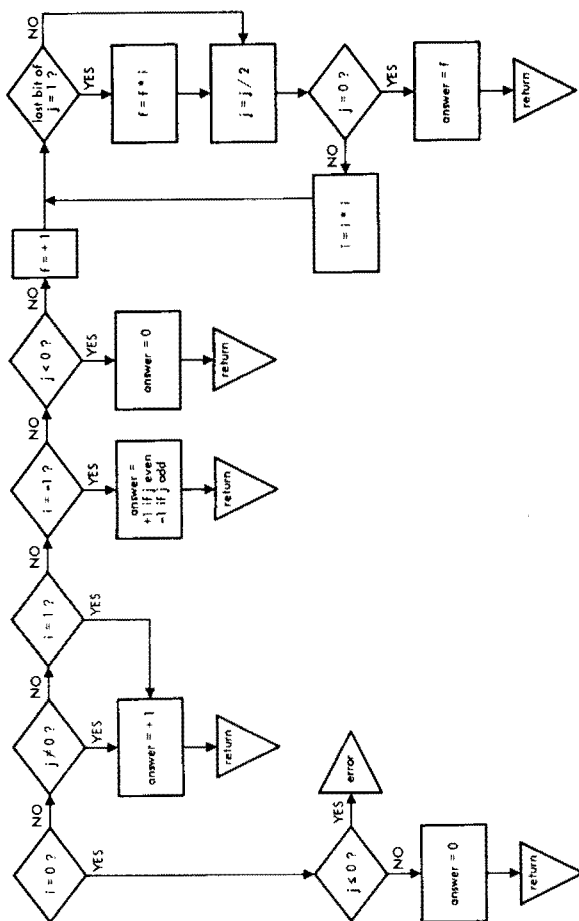
EXECUTION TIME: Minimum = .64 ms
Average = .7 ms
Maximum = .75 ms

ERROR CONDITIONS: See FAD, page 2-4.

.ITOI

PURPOSE:	To calculate i^j , where i and j have integer values.	
CALLING SEQUENCE:	JSB .ITOI DEF i DEF j	
NORMAL RETURN:	Integer result is left in A register. If error condition (below) occurs, overflow bit is set.	
STORAGE:	77 words (115 ₈)	
EXECUTION TIME:	Minimum = (1.5) (log ₂ j) (.09) ms Average = (1.5) (log ₂ j) (.12) ms Maximum = (1.5) (log ₂ j) (.18) ms	
ACCURACY:	23 bits	
ERROR CONDITIONS:	<u>Condition</u>	<u>Error Code</u>
	$i = 0, j \leq 0$	ø8 UN
	$i^j \geq 2^{23}$	ø8 OF

ALGORITHM:



MPY

PURPOSE: To multiply the two integer quantities *i* and *j*.

CALLING SEQUENCE: JSB .MPY

DEF *i* (*j* is assumed to be in the A register)

ASSEMBLY LANGUAGE: MPY *i* (*j* is assumed to be in the A register)

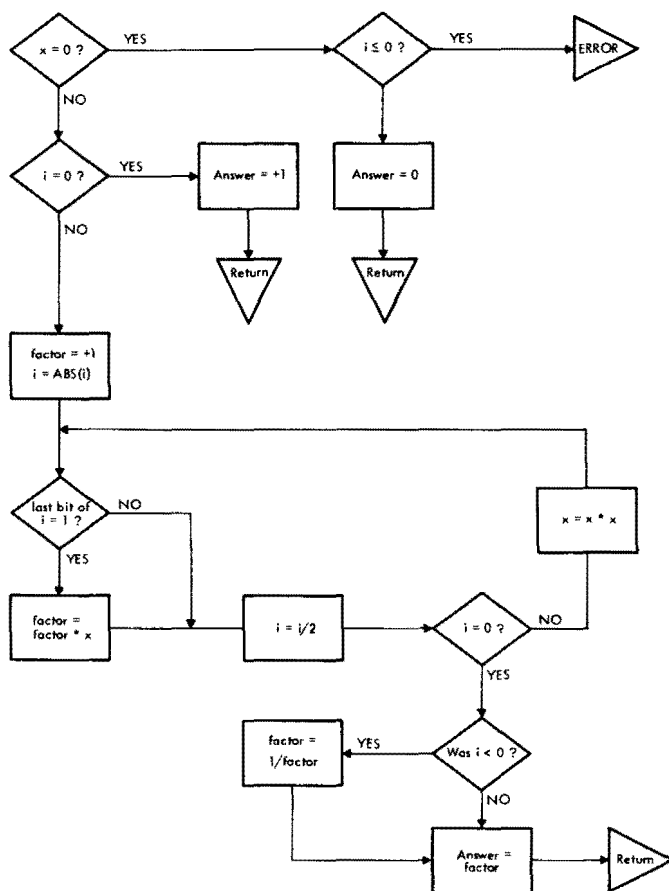
NORMAL RETURN: The result is left in the A and B registers:
B containing most significant bits,
A containing least significant bits.

STORAGE: 73 words (111₈)

EXECUTION TIME: Minimum = .09 ms
Average = .12 ms
Maximum = .15 ms

PURPOSE:	To calculate x^i , where x has a floating point and i an integer value.	
CALLING SEQUENCE:	JSB .RTOI	
	DEF x	
	DEF i	
NORMAL RETURN:	Floating point result is left in A and B registers. If error condition (below) occurs, overflow bit is set.	
STORAGE:	78 words (116 ₈)	
EXECUTION TIME:	(Approximate)	
	Minimum = (1.5) (log ₂ i) (.64) ms	
	Average = (1.5) (log ₂ i) (.70) ms	
	Maximum = (1.5) (log ₂ i) (.75) ms	
ACCURACY:	The only possibility of inaccuracy is that introduced by roundoff in the FMP or the FDV routine if i < 0.	
	x^i gives the same result as the expression:	
	$\underbrace{x*x*x*...*x}_{i \text{ times}}$	or $\underbrace{1/x*x*x*...*x}_{i \text{ times}}$
ERROR CONDITIONS:	<u>Condition</u>	<u>Error Code</u>
	$x=0, \quad i \leq 0$	Ø6 UN
	$x^{ i } > 2^{128}$	(floating point overflow)

ALGORITHM:



.RTOR

PURPOSE: To calculate x^y , where x and y have floating point values.

CALLING SEQUENCE: JSB .RTOR
DEF x
DEF y

NORMAL RETURN: Floating point result is left in A and B registers.
If error condition (below) occurs, the overflow bit is set.

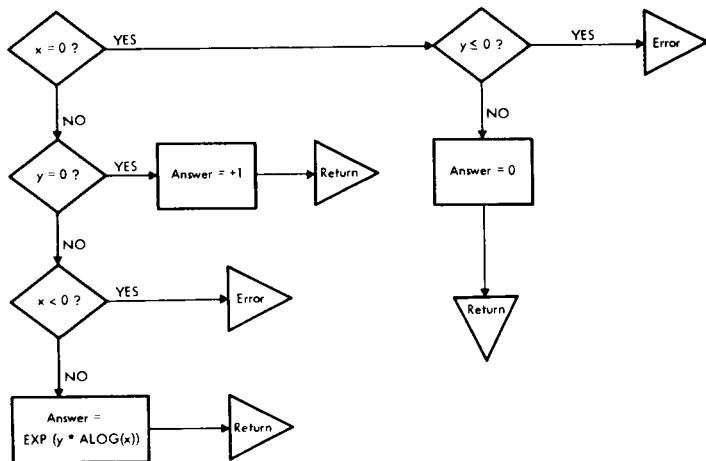
STORAGE: 46 words (56_8)

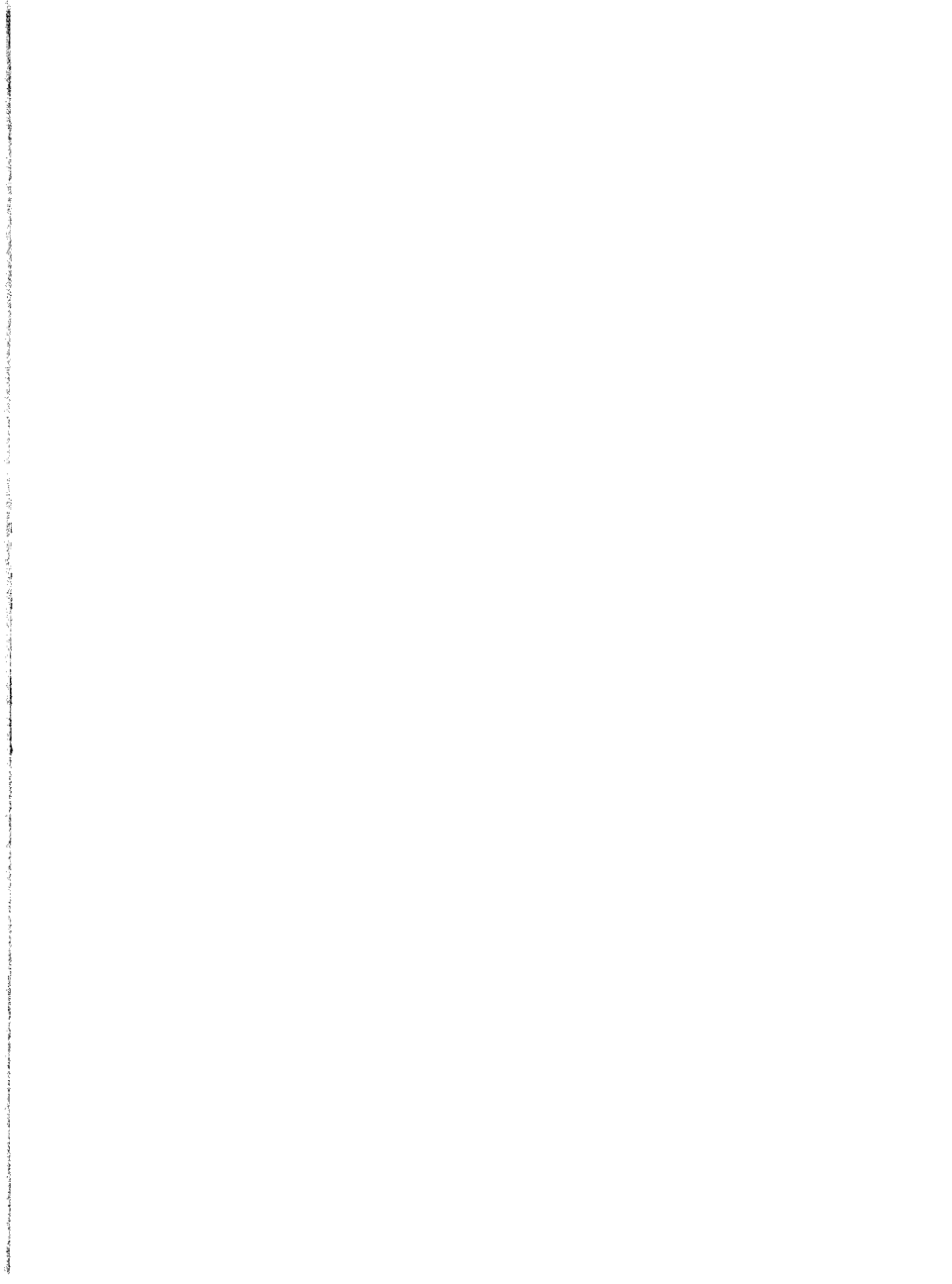
EXECUTION TIME: Minimum = 12.24 ms
Average = 15.4 ms
Maximum = 21.3 ms

ACCURACY: 22 bits

ERROR CONDITIONS:	<u>Condition</u>	<u>Error Code</u>
	$x < 0, y \leq 0$ $x = 0, y \neq 0$	Ø4 UN
	$ Y * \text{ALOG}(X) \geq 124$	Ø7 OF

ALGORITHM:





CLRIO

PURPOSE: To perform a system clear request; makes all input/output devices available for the initiation of a new operation.

CALLING SEQUENCE: JSB CLRIO
DEF *+1

NORMAL RETURN: All devices are available for new operations.

STORAGE: 5 words.

ENDIO

PURPOSE:	To delay further program execution until all current input/output operations are completed.
CALLING SEQUENCE:	JSB ENDIO DEF *+1
NORMAL RETURN:	All input/output operations are completed.
STORAGE:	7 words.

PURPOSE: To input or output data according to a specified format.

CALLING SEQUENCE: INPUT

I. Format Description

Formatted

LDA <unit>	
CLB, INB	
JSB . DIO.	
{ DEF <fmt> }	(fmt = label of ASC instruction which defines the format specifications)
{ ABS Ø }	(ABS allows for free field input)
DEF <end-of-list>	(end-of-list = The first location following the calling sequence)

Binary

LDA <unit>
CLB, INB
JSB . BIO.

II. Element Description

Real Variable

LSB . IOR.
DST <address of data>

Integer Variable

JSB . IOI.
STA <address of data>

Array

LDA <array length> (in number of elements not memory locations)

LDB <array address>
JSB . RAR. (real) or . IAR (integer)

OUTPUT**I. Format Description****Formatted**

LDA <unit>

CLB

JSB .DIO.

DEF <fmt>

(fmt = label of ASC instruction which defines format specification)

DEF <end-of-list> (end-of-list = the first location following the calling sequence)

Binary

LDA <unit>

CLB

LSB .BIO.

II. Element Description**Real Variable**

DLD <address of data>

JSB .IOR.

Integer Variable

LDA <address of data>

JSB .IOI.

Array

LDA <array length> (in number of elements not memory locations)

LDB <array address>

JSB .RAR. (real) or .IAR. (integer)

III. Terminator

JSB .DTA.

**NORMAL
RETURN:**

Data is read in or written out; control returns to the instruction immediately following the last calling sequence parameter.

**INTERNAL
CONVERSION:**

Data already in memory can be converted to or from ASCII using the following sequences.

To convert ASCII data to floating point or integer form:

```
LDA ≠ B0
CLB, INB
JSB .DIO.
DEF <buffer>      (buffer contains the ASCII infor-
                   mation to be converted)
DEF <end-of-list>
```

The remainder of the calling sequence is the same as a normal input operation. The converted data will be stored as specified in the element description.

To convert floating point or integer data to ASCII form:

```
LDA ≠ B0
CLB
JSB .DIO.
DEF <buffer>      (the converted data will be stored
                   at buffer)
DEF <fmt>
DEF <end-of-list>
```

The remainder of the calling sequence is the same as the normal output operation. The data which is to be converted is specified in the element description.

STORAGE: 1360 words (2520₈)

LEADR

PURPOSE: Produces consecutive feed frames (zeros) on punched tape to serve as leader.

**CALLING
SEQUENCE:** JSB LEADR
DEF *+3
DEF u
DEF n

The third word of the sequence defines the unit-reference number of the punched tape unit on which the leader is to be punched.

The fourth word of the sequence defines the number of inches, n, of leader which is to be punched on the tape.

**NORMAL
RETURN:** The leader is punched as requested. (An attempt to perform this operation on other than a paper tape output device will result in an illegal request code error (a halt with zeros in A-Register) which is considered to be an irrecoverable error.)

STORAGE: 27 words (33_g)

PURPOSE: To position a magnetic tape by spacing forward or backward a number of files and records.

**CALLING
SEQUENCE:**

JSB PTAPE
DEF *+4
DEF u
DEF file count
DEF record count

The third word of the sequence defines the unit-reference number of the tape unit on which a reel is to be positioned.

The fourth word defines file spacing:

- a) A positive integer for forward spacing.
- b) A negative integer for backspacing.

The fifth word defines record spacing:

- a) A positive integer for forward spacing.
- b) A negative integer for backspacing.

When spacing by record, the file mark will be read as another record mark. Thus file marks must be considered in determining the correct spacing count.

Both file and record spacing may be designated in the same call (e. g. Space backward 5 files, then forward 2 records).

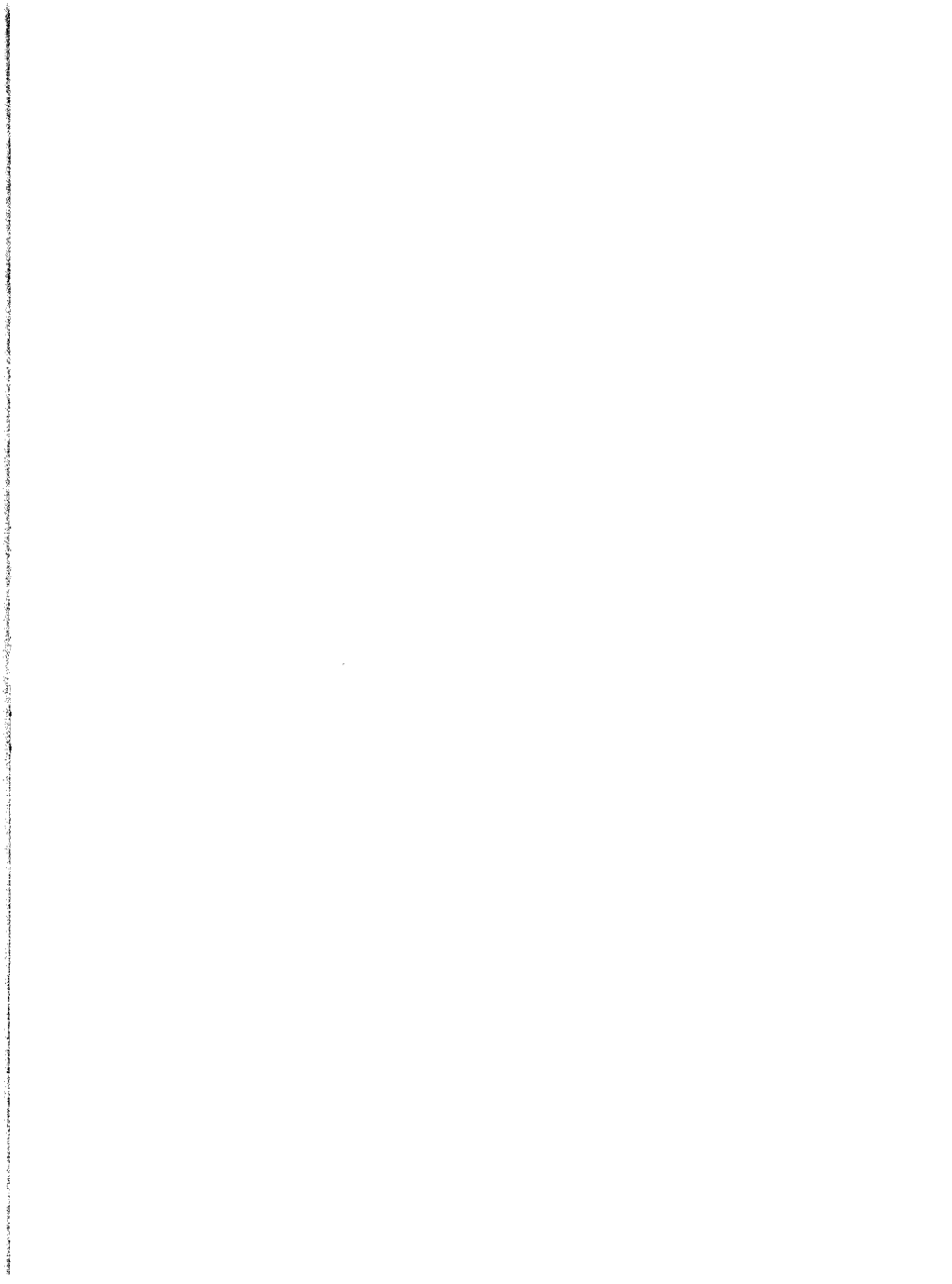
**NORMAL
RETURN:**

On completion of the spacing operation, the tape is positioned ready for reading or writing (i. e., after a backspacing operation, the read head is positioned at the beginning of a file or record). If forward spacing positions the tape beyond the End-of-Tape marker, the operator presses RUN, execution resumes at the normal return location. If backward spacing would position the tape beyond the Start-of-Tape marker, the spacing operation is terminated and execution transfers to the normal return location.

**ERROR
CONDITION:**

Condition	Message
An attempt to space beyond EOT	*EOT

STORAGE: 117 words (165₈)



CHEBY

- PURPOSE:** To evaluate the Chebyshev series represented by the table of coefficients at the argument x ; where x has a floating point value.
- CALLING SEQUENCE:** DLD x
JSB CHEBY
DEF c (c =starting address of table of floating point Chebyshev coefficients; the table is terminated by a zero word.)
- NORMAL RETURN:** Floating point result is left in A and B registers.
- STORAGE:** 57 words (71_8)
- EXECUTION TIME:** Minimum = $n(1.24)$ ms
Average = $n(1.7)$ ms (where n =number of Chebyshev coefficients in table.)
Maximum = $n(2.55)$ ms
- ACCURACY:** 22 bits
- ERROR CONDITIONS:** Possibility of floating point overflow in the case of $\text{TAN}(x)$ when x is very close to a multiple of $\pi/2$.

..FCM

PURPOSE: To complement the floating point number in the A- and B-Registers.

CALLING SEQUENCE: DLD X (x = address of floating point value)
JSB ..FCM

NORMAL RETURN: The A and B-Register will contain the complemented floating point value.

STORAGE: 6 words

.ERRR

PURPOSE: To write an error message, supplied by the calling routine, on the Standard Teleprinter Output device.

CALLING SEQUENCE: JSB .ERRR

ASC 1,xx (xx = 2 digit code identifying calling routine)

ASC 1,yy (yy = 2 letter code identifying type of error)

NORMAL RETURN: Message is printed, A and B registers are destroyed.

STORAGE: 17 words (21₈)

.FLUN

PURPOSE: To "unpack" a floating point number.

CALLING SEQUENCE: JSB .FLUN

NORMAL RETURN: Result is left with exponent in A register,
and lower part of mantissa in B register.

STORAGE: 15 words (17₈)

EXECUTION TIME: .03 ms

PURPOSE:

To calculate ENTIER(x); the greatest integer not algebraically exceeding x, where x has a floating point value.

CALLING SEQUENCE:

DLD x
 JSB .IENT
 JMP errtn (address of exit to be taken if
 EXPO(x) > 14)

NORMAL RETURN:

Integer result is left in A register.

EXECUTION TIME:

Minimum = .22 ms

Average = .38 ms

Maximum = .54 ms

STORAGE:

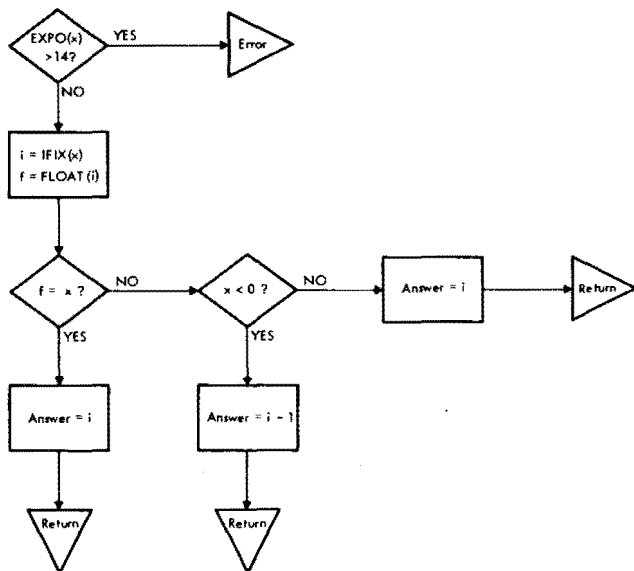
29 words (35₈)

ACCURACY:

Exact

ERROR CONDITIONS:

<u>Condition</u>	<u>Error Code</u>
EXPO(x) > 14	(depends on error routine)

ALGORITHM:

ISSW

PURPOSE: Places the contents of switch *n* of the Switch Register into the sign position (bit 15) of the A-register. The A-Register can then be tested to determine the status of the switch (0 = off; 1 = on).

CALLING SEQUENCE: LDA *n* (*n* is the number of the switch to be tested)
JSB ISSW

NORMAL RETURN: The A-Register contains the image of the Switch Register which has been rotated until switch *n* occupies the sign bit position.

STORAGE: 7 words.

PURPOSE:

To extract mantissa part of x , where x has a floating point value.

Thus $x = \text{MANT}(x) * 2^{**} \text{EXPO}(x)$.

CALLING SEQUENCE:

DLD x

JSB MANT

NORMAL RETURN:

Floating point result is left in A and B registers.

EXECUTION TIME:

.023 ms

STORAGE:

9 words (11_8)

ACCURACY:

23 bits

.PACK

PURPOSE: To transform the signed 31-bit mantissa contained in A and B registers to normalized floating point format.

CALLING SEQUENCE: JSB .PACK
BSS 1 (Contains exponent portion of floating point number)

NORMAL RETURN: Floating point result is left in A and B registers.

STORAGE: 69 words (105_8)

EXECUTION TIME: .1 to .15 ms

PURPOSE: To calculate $x (2^n)$, where x has a floating point value and n is an integer.

CALLING SEQUENCE: DLD x
JSB PWR2
DEC n

NORMAL RETURN: Floating point result is left in A and B registers.

EXECUTION TIME: .075 ms

STORAGE: 20 words (24_8)

ALGORITHM: Exponent of x is increased by n .

ACCURACY: 23 bits

ERROR CONDITIONS: None. Calling routine should check for possibility of overflow.



..DLC

- PURPOSE:** To load and complement a floating point quantity.
- CALLING SEQUENCE:** JSB ..DLC
DEF X (X = address of the two-word quantity to be complemented)
- NORMAL RETURN:** The original contents of the A & B-Registers will be lost; the complemented quantity will be loaded into A & B-Registers.
- STORAGE:** 10 words (12₈)

ENTIE

PURPOSE: To calculate the greatest integer not algebraically exceeding a floating point value.

CALLING SEQUENCE: DLD X (X = address of floating-point value)
JSB ENTIE

NORMAL RETURN: The A-Register will contain the sign; the B-Register will contain the maximum integer.

STORAGE: 33 words (41₈)

.ENTR

PURPOSE: To transfer the actual parameters of a FORTRAN function to the function or subroutine being entered.

CALLING SEQUENCE: JSB .ENTR
DEF a (a = First location of area used to store argument address)

NORMAL RETURN: Stores parameters.

STORAGE: 38 words (46₈)

.GOTO

PURPOSE: To perform translation of a FORTRAN computed GO TO statement, GO TO ($k_1, k_2, \dots k_n$), i.

CALLING SEQUENCE: JSB .GOTO
DEF (return address)
DEF (address of switch index j)
DEF (address of label k_1)
.
.
DEF (address of label k_n)

NORMAL RETURN: Control will be switched to the correct statement.
(See page 5-1 of FORTRAN Programmer's Reference Manual)

STORAGE: 19 words (28_8)

PURPOSE: To take the logical product of i and j and store the result in the A-Register.

CALLING SEQUENCE: JSB LAND
DEF i (address of first argument)
DEF j (address of second argument)

NORMAL RETURN: The logical product will be left in the A-Register.

STORAGE: 8 words (10_8)

INDEX

PURPOSE: To produce the absolute address of an array element where the array is described elsewhere in an array table.

CALLING SEQUENCE: .INDA and .INDR are used by the ALGOL compiler in order to access array elements. .INDA produces the absolute address of an array element, whereas .INDR produces the value stored in that location. The calling sequence to this routine is:

```
JSB   .INDA or .INDR
DEF   array table
ABS   -number of indices
DEF   index 1
....
DEF   index n
```

The array table for a given array has the form:

```
TABLE ABS   number of indices (+ = real, - = integer)
      ABS   size of 1st dim
      ABS   -lower bound of 1st dim
      ....
      ABS   size of last DIM
      ABS   -lower bound of last dim
```

NORMAL RETURN:

```
.INDA:  A = address of array element
.INDR:  A = value if integer
        A & B = value if real
```

ERROR RETURN: Computer halts with A = address of JSB .INDA and prints INDEX? on teleprinter. When run is pushed, routine returns to the program with value = 0.

STORAGE: 81 words (121₈)

PURPOSE: To take logical inclusive - or of i and j and return result in the A-Register.

CALLING SEQUENCE: JSB IOR
DEF i (i = address of first argument)
DEF j (j = address of second argument)

NORMAL RETURN: The A-Register contains the result.

STORAGE: 7 words

.MAP.

PURPOSE: This routine obtains the address of an element of a two dimensional array.

CALLING SEQUENCE:

```
JSB    .MAP.  
DEF    (Base address of array)  
DEF    (address of first dimension)  
DEF    (address of second dimension)  
OCT    (First dimension of array, two's  
        complement if integer array)
```

NORMAL RETURN: The A-Register contains the address.

STORAGE: 31 words (37₈)

PURPOSE: Performs memory allocation for buffering.
Requests may be made to allocate and release buffers from the memory available after loading.

Allocate

CALLING SEQUENCE: JSB .ALC.
DEC X (X = number of words needed)

NORMAL RETURN: In the A-Register is FWA buffer; in the B-Register, the number of words allocated.

Release Buffer

CALLING SEQUENCE: JSB .RTN.
DEF (address of first word of buffer)
DEF (number of words allocated)

NORMAL RETURN: Registers unchanged; buffers released.

NOTE: To release all storage allocated, insert the following before the release calling sequence.

CLA
STA .CLR.

STORAGE: 214 words (326₈)

OVF

PURPOSE:	Returns true value (sign bit = 1) in the A-Register if overflow bit is set.
CALLING SEQUENCE:	(Number to be checked is in A-Register) JSB OVF
NORMAL RETURN:	A-Register is set to zero if no overflow has occurred; if an overflow condition is sensed the A-Register is set to 177777B.
STORAGE:	4 words

PURPOSE: To print PAUSE on the Teleprinter and halt the computer with i in the A-Register.

CALLING SEQUENCE: LDA i
JSB .PAUS

NORMAL RETURN: The A-Register will display i when the computer halts; PAUSE will be printed on the teleprinter.

STORAGE: 18 words (22₈)

.PRAM

PURPOSE: Processes and sets parameter values and/or addresses.

**CALLING
SEQUENCE:**

JSB .PRAM

(code words)

1st parameter address or value (real = 2 words)

2nd parameter address or value (real = 2 words)

.....

last parameter address or value

Code Words:

- 1st: upper 6 bits contains no. of parameters
lower 10 bits contain flags for each of 1st 5
parameters in pairs of bits
upper = 1, parameter is by value.
lower = 1, parameter value is real.

If upper bit = 0, the linkage contains the parameter address (ignore lower).

If upper bit = 1, the linkage contains the parameter value.

If both bits = 1, the value is real and the value will need 2 words in the linkage.

Subsequent code words:

contain 8 pairs of bits for 8 parameters, maximum of 7 code words (52 parameters).

**NORMAL
RETURN:**

Parameters will be processed.

STORAGE:

72 words (110₈)

PURPOSE: To round a real value to the nearest integer value (used with ENTIE). Ties are decided by using the algebraically larger integer.

CALLING SEQUENCE: JSB .RND

NORMAL RETURN: The result will be left in the A-Register.

STORAGE: 33 words (41₈)

.STOP

PURPOSE:

To print STOP on the Teleprinter and halt the computer with i in the A-Register.

CALLING SEQUENCE:

LDA i
JSB .STOP

NORMAL RETURN:

STOP will be printed on the teleprinter; the computer will halt and display i in the A-Register. The subroutine will repeat itself when the operator pushes RUN.

STORAGE:

17 words (21₈)

PURPOSE: To switch control to one of a sequence of labels provided the sequence number of the label is in the A-Register.

CALLING SEQUENCE: LDA i (i = value of switch)
JSB .SWCH
OCT (label count)

NORMAL RETURN: The program will jump to the indicated label.

STORAGE: 15 words (17₈)

.TAPE

PURPOSE: To call .IOC, with parameters generated by FORTRAN auxiliary Input/Output statements.

CALLING SEQUENCE: LDA 30xyy₈
JSB .TAPE

(The 30xyy₈ is the .IOC, request code.

x = 4 for REWIND
x = 2 for BACKSPACE
x = 1 for END FILE
yy = logical unit number)

NORMAL RETURN: A tape operation for a FORTRAN compiled program will be initiated.

STORAGE: 5 words

SUMMARY OF ERROR CONDITIONS AND CODES A

Error codes are produced on the Standard Teleprinter Output device at object program execution time; the general format is:

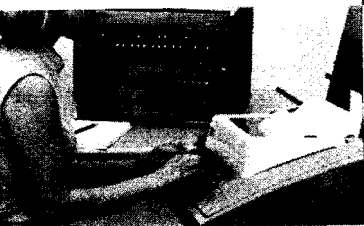
xx yy where xx is a two-digit code identifying the routine issuing the code, and yy is a two-letter mnemonic identifying the type of error. UN = underflow, OF = overflow, OR = out of range.

<u>Error Code</u>	<u>Subroutine</u>	<u>Condition</u>
02 UN	ALOG	$x \leq 0$
03 UN	SQRT	$x < 0$
04 UN	.RTOR	$x = 0, y \neq 0$ $x < 0, y \leq 0$
05 OR	SIN COS	$ABS(x) > 2^{14}$
06 UN	.RTOI	$x = 0, i \leq 0$
07 OF	.RTOR EXP	$x^{ y }$ out of range $ABS(x) * \log_2 e > 124$
08 OF	.ITOI	i^j out of range
08 UN	.ITOI	$i = 0, j \leq 0$
09 OR	TAN	$x > 2^{14}$

Overflow may be caused in the TAN(x) routine when tan(x) is out of range; in the .RTOI routine when $x^{**}ABS(i)$ is out of range.

ALPHABETICAL LISTING OF SUBROUTINES B ON THE 2018 LIBRARY TAPE

ABS, 1-1	.ERRR, 4-3	ISSW, 4-6	.PRAM, 5-12
ALOG, 1-2	FADSB, 2-4, 2-5	.IENT, 4-5	.RND, 5-2
ATAN, 1-3	.FDV, 2-6	.IOR, 5-7	.RTOI, 2-10
CHEBY, 4-1	FLOAT, 1-6	.ITOI, 2-10	.RTOR, 2-12
CLR10, 3-1	.FMP, 2-7	LEADR, 3-6	SIGN, 1-10
COS, 1-4	FRMTR, 3-3	MANT, 4-7	SIN, 1-11
DIV, 2-1	.FLUN, 4-4	MEMRY, 5-9	SQRT, 1-12
DLDST, 2-2	..FCM, 4-2	.MPY, 2-9	.SCN
D. 43	GOTO, 5-4	.MAP., 5-8	.STOP, 5-14
..DLC, 5-1	IABS, 1-7	OVF, 5-10	.SWCH, 5-15
ENDIO, 3-2	IAND, 5-5	PTAPE, 3-7	TAN, 1-13
ENTIE, 5-2	IFIX, 1-8	PWR2, 4-9	TANH, 1-14
EXP, 1-5	INDEX, 5-6 (INDA, INDR)	.PACK, 4-8	.TAPE, 5-16
.ENTR, 5-3	ISIGN, 1-9	.PAUS, 5-11	.2018



BASIC Language Reference Manual

The development of the BASIC Language and the original version of this manual on BASIC were supported in part by the National Science Foundation under terms of Grant NSF GE 3864 to Dartmouth College. Under this grant, Dartmouth College, under the direction of Professors John G. Kemeny and Thomas E. Kurtz, developed the BASIC Language and a compiler for operation under a time-sharing system in use at Dartmouth.

This printing of this manual by Hewlett-Packard does not necessarily constitute endorsement of Hewlett-Packard products by Dartmouth College.

- © Copyright 1968 by Trustees of Dartmouth College. Reproduced with the permission of the Trustees of Dartmouth College.

CONTENTS

INTRODUCTION	v
CHAPTER 1 A BASIC PRIMER	1-1
1.1 An Example	1-1
1.2 Formulas	1-4
1.3 Loops	1-7
1.4 Arrays	1-10
1.5 Use of The System	1-12
1.6 Errors and Debugging	1-13
1.7 Summary of Elementary BASIC Statements	1-17
1.7.1 LET	1-17
1.7.2 READ and DATA	1-17
1.7.3 PRINT	1-18
1.7.4 GO TO	1-19
1.7.5 IF - THEN	1-19
1.7.6 FOR and NEXT	1-19
1.7.7 DIM	1-20
1.7.8 END	1-20
CHAPTER 2 ADVANCED BASIC	2-1
2.1 More About PRINT	2-1
2.2 Functions and DEF	2-4
2.2.1 INT	2-4
2.2.2 RND	2-4
2.2.3 SGN	2-6
2.2.4 DEF	2-6
2.3 GOSUB AND RETURN	2-6
2.4 INPUT	2-7
2.5 Some Miscellaneous Statements	2-8
2.5.1 STOP	2-8
2.5.2 REM	2-8
2.5.3 RESTORE	2-9
2.5.4 COM	2-9
2.6 Matrices	2-10
2.7 Error Codes	2-16
APPENDIX A USING THE HP BASIC SYSTEM	A-1
APPENDIX B CALL AND WAIT STATEMENTS	B-1
APPENDIX C LOGICAL OPERATORS IN THE BASIC LANGUAGE	C-1
APPENDIX D BASIC SYNTAX IN BACKUS NORMAL FORM	D-1
APPENDIX E PREPARE BASIC SYSTEM PROGRAM (PBS)	E-1

INTRODUCTION

This publication is a reference manual for using the HP BASIC System with HP computers. It includes both the elements of the language and the information required to operate the system on the computer. The minimum configuration on which the HP BASIC Compiler will run is:

HP 2116B, 2115A, or 2114A Computer with 8196 words of memory
HP 2752A Teleprinter

WHAT IS A PROGRAM ?

A program is a set of directions, or a recipe, that is used to tell a computer how to provide an answer to some problem. It usually starts with the given data as the ingredients, contains a set of instructions to be performed or carried out in a certain order, and ends up with a set of answers as the cake. And, as with ordinary cakes, if you make a mistake in your program, you will end up with something else - perhaps hash!

Any program must fulfill two requirements before it can be carried out. The first is that it must be presented in a language that is understood by the "computer". If the program is a set of instructions for solving a system of linear equations and the "computer" is an English-speaking person, the program will be presented in some combination of mathematical notation and English. If the "computer" is a French-speaking person, the program must be in his language; and if the "computer" is a high-speed digital computer, the program must be presented in a language which the computer "understands".

The second requirement for all programs is that they must be completely and precisely stated. This requirement is crucial when dealing with a digital computer which has no ability to infer what you mean - it does what you tell it to do, not what you meant to tell it.

We are, of course, talking about programs which provide numerical answers to numerical problems. It is easy to program in the English language, but such a program poses great difficulties for the computer because English is rich with ambiguities and redundancies, those qualities which make poetry possible but computing impossible. Instead, you present your program in a language which resembles ordinary mathematical notation, which has a simple vocabulary and grammar, and which permits a complete and precise specification of your program. The language you will use is BASIC which is, at the same time, precise, simple and easy to understand.

A first introduction to writing a BASIC program is given in Chapter 1. This chapter includes all that you will need to know to write a wide variety of useful and interesting programs. Chapter 2 deals with more advanced computer techniques, and the Appendices contain a variety of reference materials.

1.1 AN EXAMPLE

The following example is a complete BASIC program for solving a system of two simultaneous linear equations in two variables:

$$ax + by = c$$

$$dx + ey = f$$

And then solving two different systems, each differing from this system only in the constants c and f .

You should be able to solve this system, if $ae - bd$ is not equal to 0, to find that:

$$x = \frac{ce - bf}{ae - bd} \quad \text{and} \quad y = \frac{af - cd}{ae - bd}$$

If $ae - bd = 0$, there is either no solution or there are infinitely many, but there is no unique solution. If you are rusty on solving such systems, take our word for it that this is correct. For now, we want you to understand the BASIC program for solving this system.

Study this example carefully - in most cases the purpose of each line in the program is self-evident - and then read the commentary and explanation.

```
10  READ A, B, D, E
15  LET G = A * E - B * D
20  IF G = 0 THEN 65
30  READ C, F
37  LET X = (C * E - B * F) / G
42  LET Y = (A * F - C * D) / G
55  PRINT X, Y
60  GO TO 30
65  PRINT "NO UNIQUE SOLUTION"
70  DATA 1, 2, 4
80  DATA 2, -7, 5
85  DATA 1, 3, 4, -7
90  END
```

We immediately observe several things about this sample program. First, we see that the program uses only capital letters, since the teletypewriter has only capital letters.

A second observation is that each line of the program begins with a number. These numbers are called line numbers and serve to identify the lines, each of which is called a statement. Thus, a program is made up of statements, most of which are instructions to the computer. Line numbers also serve to specify the order in which the statements are to be performed by the computer. This means that you may type your program in any order. Before the program is run, the computer sorts out and edits the program, putting the statements into the order specified by their line numbers. (This editing process facilitates the correcting and changing of programs, as we shall explain later.)

A third observation is that each statement starts, after its line number, with an English word. This word denotes the type of the statement. There are several types of statements in BASIC, nine of which are discussed in this chapter. Seven of these nine appear in the sample program of this section.

A fourth observation, not at all obvious from the program, is that spaces have no significance in BASIC, except in messages enclosed in quotation marks which are to be printed out, as in line number 65 above. Thus, spaces may be used, or not used, at will to "pretty up" a program and make it more readable. Statement 10 could have been typed as 10READA, B, D, E and statement 15 as 15LETG=A*B*D.

With this preface, let us go through the example, step by step. The first statement, 10, is a READ statement. It must be accompanied by one or more DATA statements. When the computer encounters a READ statement while executing your program, it will cause the variables listed after the READ to be given values according to the next available numbers in the DATA statements. In the example, we read A in statement 10 and assign the value 1 to it from statement 70 and, similarly with B and 2, and with D and 4. At this point, we have exhausted the available data in statement 70, but there is more in statement 80, and we pick up from it the number 2 to be assigned to E.

We next go to statement 15, which is a LET statement, and first encounter a formula to be evaluated. (The asterisk "*" is obviously used to denote multiplication.) In this statement we direct the computer to compute the value of $AE - BD$, and to call the result G. In general, a LET statement directs the computer to set a variable equal to the formula on the right side of the equals sign. We know that if G is equal to zero, the system has no unique solution. Therefore, we next ask, in line 20, if G is equal to zero. If the computer discovers a "yes" answer to the question, it is directed to go to line 65, where it prints "NO UNIQUE SOLUTION". From this point, it would go to the next statement. But lines 70, 80, and 85 give it no instructions, since DATA statements are not "executed", and it then goes to line 90 which tells it to "END" the program.

If the answer to the question "Is G equal to zero?" is "no", as it is in this example, the computer goes on to the next statement, in this case 30. (Thus, an IF-THEN tells the computer where to go if the "IF" condition is met, but to go on to the next statement if it is not met.) The computer is now directed to read the next two entries from the DATA statements, -7 and 5, (both are in statement 80) and to assign them to C and F respectively. The computer is now ready to solve the system

$$x + 2y = -7$$

$$4x + 2y = 5.$$

In statements 37 and 42, we direct the computer to compute the value of X and Y according to the formulas provided. Note that we must use parentheses to indicate that $CE - BF$ is divided by G; without parentheses, only BF would be divided by G and the computer would let $X = CE - \frac{BF}{G}$.

The computer is told to print the two values computed, that of X and that of Y, in line 55. Having done this, it moves on to line 60 where it is directed back to line 30. If there are additional numbers in the DATA statements, as there are here in 85, the computer is told in line 30 to take the next one and assign it to C, and the one after that to F. Thus, the computer is now ready to solve the system

$$x + 2y = 1$$

$$4x + 2y = 3.$$

As before, it finds the solution in 37 and 42 and prints them out in 55, and then is directed in 60 to go back to 30.

In line 30 the computer reads two more values, 4 and -7, which it finds in line 85. It then proceeds to solve the system

$$x + 2y = 4$$

$$4x + 2y = -7$$

and to print out the solutions. It is directed back again to 30, but there are no more pairs of numbers available for C and F in the DATA statements. The computer then informs you that it is out of data, printing on the paper in your teletypewriter "ERROR 56 IN LINE 30"

For a moment, let us look at the importance of the various statements. For example, what would have happened if we had omitted line number 55? The answer is simple: the computer would have solved the three systems and then told us when it was out of data. However, since it was not asked to tell us (PRINT) its answers, it would not do it, and the solutions would be the computer's secret. What would have happened if we had left out line 20? In this problem just solved nothing would have happened. But, if G were equal to zero, we would have set the computer the impossible task of dividing by zero, and it would tell us so, printing "ERROR 69 IN LINE 37". Had we left out statement 60, the computer would have solved the first system, printed out the values of X and Y, and then gone on to line 65 where it would be directed to print "NO UNIQUE SOLUTION". It would do this and then stop.

One very natural question arises from the seemingly arbitrary numbering of the statements: why this selection of line numbers? The answer is that the particular choice of line numbers is arbitrary, as long as the statements are numbered in the order which we want the machine to follow in executing the program. We could have numbered the statements 1, 2, 3, ..., 13, although we do not recommend this numbering. We would normally number the statements 10, 20, 30, ..., 130. We put the numbers such a distance apart so that we can later insert additional statements if we find that we have forgotten them in writing the program originally. Thus, if we find that we have left out two statements between those numbered 40 and 50, we can give them any two numbers between 40 and 50 - say 44 and 46; and in the editing and sorting process, the computer will put them in their proper place.

Another question arises from the seemingly arbitrary placing of the elements of data in the DATA statements: why place them as they have been in the sample program? Here again the choice is arbitrary and we need only put the numbers in the order that we want them read (the first for A, the second for B, the third for D, the fourth for E, the fifth for C, the sixth for F, the seventh for the next C, etc.) In place of the three statements numbered 70, 80, and 85, we could have put

```
75 DATA 1, 2, 4, 2, -7, 5, 1, 3, 4, -7
```

or we could have written, perhaps more naturally,

```
70 DATA 1, 2, 4, 2
```

```
75 DATA -7, 5
```

```
80 DATA 1, 3
```

```
85 DATA 4, -7
```

to indicate that the coefficients appear in the first data statement and the various pairs of right-hand constants appear in the subsequent statements.

The program and the resulting run is shown below exactly as it appears on the teletypewriter.

```
READY
10 READ A, B, D, E
15 LET G = A * E - B * D
20 IF G = 0 THEN 65
30 READ C, F
37 LET X = ( C * E - B * F ) / G
42 LET Y = ( A * F - C * D ) / G
55 PRINT X, Y
60 GO TO 30
65 PRINT "NO UNIQUE SOLUTION"
70 DATA 1, 2, 4
80 DATA 2, -7, 5
85 DATA 1, 3, 4, -7
90 END
RUN
  4                -5.5
.666667           .166667
-3.66667          3.83333
ERROR 56 IN LINE 30
```

After typing the program, we type RUN followed by a CARRIAGE RETURN. Up to this point the computer stores the program and checks the form of the statements. It is this command which directs the computer to execute your program. The message out-of-data error code here may be ignored. However, in some cases it indicates an error in the program: for more details see Sec. 1.7.2.

1.2 FORMULAS

The computer can perform a great many operations; it can add, subtract, multiply, divide, extract square roots, raise a number to a power, and find the sine of a num-

ber (on an angle measured in radians), etc. - and we shall now learn how to tell the computer to perform these various operations and to perform them in the order that we want them done.

The computer performs its primary function (that of computation) by evaluating formulas which are supplied in a program. These formulas are very similar to those used in standard mathematical calculation, with the exception that all BASIC formulas must be written on a single line. Five arithmetic operations can be used to write a formula, and these are listed in the following table:

Symbol	Example	Meaning
+	$A + B$	Addition (add B to A)
-	$A - B$	Subtraction (subtract B from A)
*	$A * B$	Multiplication (multiply B by A)
/	A / B	Division (divide A by B)
↑	$X \uparrow 2$	Raise to the power (find X^2)

We must be careful with parentheses to make sure that we group together those things which we want together. We must also understand the order in which the computer does its work. For example, if we type $A + B * C \uparrow D$, the computer will first raise C to the power D, multiply this result by B, and then add A to the resulting product. This is the same convention as is usual for $A + B * C^D$. If this is not the order intended, then we must use parentheses to indicate a different order. For example, if it is the product of B and C that we want raised to the power D, we must write $A + (B * C) \uparrow D$; or, if we want to multiply A + B by C to the power D, we write $(A + B) * C \uparrow D$. We could even add A to B, multiply their sum by C, and raise the product to the power D by writing $((A + B) * C) \uparrow D$. The order of priorities is summarized in the following rules:

1. The formula inside parentheses is computed before the parenthesized quantity is used in further computations.
2. In the absence of parentheses in a formula involving addition, multiplication, and the raising of a number to the power, the computer first raises the number to the power, then performs the multiplication, and the addition comes last. Division has the same priority as multiplication, and subtraction the same as addition.
3. In the absence of parentheses in a formula involving operations of the same priority, the operations are performed from left to right.

These rules are illustrated in the previous example. The rules also tell us that the computer, faced with $A - B - C$, will (as usual) subtract B from A and then C from their difference; faced with $A/B/C$, it will divide A by B and that quotient by C. Given $A \uparrow B \uparrow C$, the computer will raise the number A to the power B and take the resulting number and raise it to the power C. If there is any question in your mind about the priority, put in more parentheses to eliminate possible ambiguities.

In addition to these five arithmetic operations, the computer can evaluate several mathematical functions. These functions are given special 3-letter English names, as the following list shows:

Functions	Interpretation	
SIN (X)	Find the sine of X	} X interpreted as a number, or as an angle measured in radians
COS (X)	Find the cosine of X	
TAN (X)	Find the tangent of X	
ATN (X)	Find the arctangent of X	
EXP (X)	Find e^X	
LOG (X)	Find the natural logarithm of X ($\ln X$)	
ABS (X)	Find the absolute value of X ($ X $)	
SQR (X)	Find the square root of X (\sqrt{X})	

Three other functions are also available in BASIC: INT, RND, and SGN; these are reserved for explanation in Chapter 2. In place of X, we may substitute any formula or any number in parentheses following any of these formulas. For example, we may ask the computer to find $\sqrt{4 + X^3}$ by writing `SQR (4 + X^3)`, or the arctangent of $3X - 2e^X + 8$ by writing `ATN (3 * X - 2 * EXP (X) + 8)`.

If, sitting at the teletypewriter, you need the value of $(\frac{5}{6})^{17}$ you can write the two-line program

```
10 PRINT (5/6) ^ 17
20 END
```

and the computer will find the decimal form of this number and print it out in less than it took you to type the program.

Since we have mentioned numbers and variables, we should be sure that we understand how to write numbers for the computer and what variables are allowed. A number may be positive or negative and it may contain up to approximately seven significant digits, but it must be expressed in decimal form. For example, all of the following are numbers in BASIC: 2, -3.675, 1234567, -.7654321, and 483.4156. The following are not numbers in BASIC: $14/3$ and $\sqrt{7}$. We may ask the computer to find the decimal expansion of $14/3$ or $\sqrt{7}$, and to do something with the resulting number, but we may not include either in a list of DATA. We gain further flexibility by use of the letter E, which stands for "times ten to the power". Thus, we may write .001234567 in a form acceptable to the computer in any of several forms: .1234567E-2 or 1234567E-9 or 1234.56789E-6. We may write ten million as 1E7 (or $1E + 7$) and 1965 as 1.965E3 (or $1.965E + 3$). We do not write E7 as a number, but must write 1E7 to indicate that it is 1 that is multiplied by 10^7 . All numbers are represented in the computer by two 16-bit words. The exponent and its sign are eight bits; the fraction and its sign are twenty-four bits. They have a range in magnitude of approximately 10^{-38} to 10^{38} and may assume positive, negative or zero values. If a fraction is negative the number is stored in two's complement form. A zero value is stored as all zero bits. If a number is too large, the com-

puter prints "ERROR 65 IN LINE nn" and replaces it with the largest representable number of the same sign. If a number is too small, the computer prints "ERROR 66 IN LINE nn" and replaces it with zero.

A numerical variable in BASIC is denoted by any letter, or by any letter followed by a single digit. Thus, the computer will interpret E7 as a variable, along with A, X, N5, I0, and O1. A variable in BASIC stands for a number, usually one that is not known to the programmer at the time the program was written. Variables are given or assigned values by LET, READ, or INPUT statements. The value so assigned will not change until the next time a LET, READ, or INPUT statement is encountered with a value for that variable. However, all variables are set to "undefined" before a RUN. Thus, it is always necessary to assign a value to a variable before using the variable in a computation. Failure to do so will cause the computer to print "ERROR 50 IN LINE nn".

Six other mathematical symbols are provided for in BASIC, symbols of relation, and these are used in IF-THEN statements where it is necessary to compare values. An example of the use of these relation symbols was given in the sample program in Section 1.

Any of the following six standard relations may be used:

<u>Symbol</u>	<u>Example</u>	<u>Meaning</u>
=	A = B	Is equal to (A is equal to B)
<	A < B	Is less than (A is less than B)
<=	A <= B	Is less than or equal to (A is less than or equal to B)
>	A > B	Is greater than (A is greater than B)
>=	A >= B	Is greater than or equal to (A is greater than or equal to B)
#	A # B	Is not equal to (A is not equal to B)

1.3 LOOPS

We are frequently interested in writing a program in which one or more portions are performed not just once but a number of times, perhaps with slight changes each time. In order to write the simplest program, the one in which this portion to be repeated is written just once, we use the programming device known as a loop.

The programs which use loops can, perhaps, be best illustrated and explained by two programs for the simple task of printing out a table of the first 100 positive integers together with the square root of each. Without a loop, our program would be 101 lines long and read:

```
10 PRINT 1, SQR (1)
20 PRINT 2, SQR (2)
30 PRINT 3, SQR (3)
. . . . .
```

```

990    PRINT 99, SQR (99)
1000   PRINT 100, SQR (100)
1010   END

```

With the following program, using one type of loop, we can obtain the same table with far fewer lines of instruction, 5 instead of 101:

```

10    LET X = 1
20    PRINT X, SQR (X)
30    LET X = X + 1
40    IF X <= 100 THEN 20
50    END

```

Statement 10 gives the value of 1 to X and "initializes" the loop. In line 20 both 1 and its square root are printed. Then, in line 30, X is increased by 1, to 2. Line 40 asks whether X is less than or equal to 100; an affirmative answer directs the computer back to line 20. Here it prints 2 and $\sqrt{2}$, and goes to 30. Again X is increased by 1, this time to 3, and at 40 it goes back to 20. This process is repeated line 20 (print 3 and $\sqrt{3}$), line 30 (X = 4), line 40 (since $4 \leq 100$ go back to line 20), etc - until the loop has been traversed 100 times. Then, after it has printed 100 and its square root, X becomes 101. The computer now receives a negative answer to the question in line 40 (X is greater than 100, not less than or equal to it), does not return to 20 but moves on to line 50, and ends the program. All loops contain four characteristics; initialization (line 10), the body (line 20), modification (line 30), and an exit test (line 40). Because loops are so important and because loops of the type just illustrated arise so often, BASIC provides two statements to specify a loop even more simply. They are the FOR and NEXT statements, and their use is illustrated in the program:

```

10    FOR X = 1 TO 100
20    PRINT X, SQR (X)
30    NEXT X
50    END

```

In line 10, X is set equal to 1, and a test is set up, like that of line 40 above. Line 30 carries out two tasks: X is increased by 1, and the test is carried out to determine whether to go back to 20 or go on. Thus lines 10 and 30 take the place of lines 10, 30, and 40 in the previous program - and they are easier to use.

Note that the value of X is increased by 1 each time we go through the loop. If we wanted a different increase, we could specify it by writing

```

10    FOR X = 1 TO 100 STEP 5

```

and the computer would assign 1 to X on the first time through the loop, 6 to X on the second time through, 11 on the third time, and 96 on the last time. Another step of 5 would take X beyond 100, so the program would proceed to the end after printing 96 and its square root. The STEP may be positive or negative, and we could have obtained the first table, printed in reverse order, by writing line 10 as

10 FOR X = 100 TO 1 STEP -1

In the absence of a STEP clause, a step size of + 1 is assumed.

More complicated FOR statements are allowed. The initial value, the final value, and the step size may all be formulas of any complexity. For example, if N and Z have been specified earlier in the program, we could write

FOR X = N + 7 * Z TO (Z-N) / 3 STEP (N-4 * Z) / 10

For a positive step-size, the loop continues as long as the control variable is algebraically less than or equal to the final value. For a negative step-size, the loop continues as long as the control variable is greater than or equal to the final value.

If the initial value is greater than the final value (less than for negative step-size), then the body of the loop will not be performed at all, but the computer will immediately pass to the statement following the NEXT. As an example, the following program for adding up the first n integers will give the correct result 0 when n is 0.

```
10       READ N
20       LET S = 0
30       FOR K = 1 TO N
40       LET S = S + K
50       NEXT K
60       PRINT S
70       GO TO 10
90       DATA 3, 10, 0
99       END
```

It is often useful to have loops within loops. These are called nested loops and can be expressed with FOR and NEXT statements. However, they must actually be nested and must not cross, as the following skeleton examples illustrate:

ALLOWED

```
FOR X
  FOR Y
    NEXT Y
  NEXT X
```

NOT ALLOWED

```
FOR X
  FOR Y
    NEXT X
  NEXT Y
```

ALLOWED

```
FOR X
  FOR Y
    FOR Z
      NEXT Z
    FOR W
      NEXT W
    NEXT Y
  FOR Z
    NEXT Z
  NEXT X
```

1.4 ARRAYS

In addition to the ordinary variables used by BASIC, there are variables which can be used to designate the elements of an array. These are used where we might ordinarily use a subscript or a double subscript, for example the coefficients of a polynomial $[a_0, a_1, a_2, \dots]$ or the elements of a matrix $[b_{i,j}]$. The variables which we use in BASIC consist of a single letter, which we call the name of the array, followed by the subscripts in brackets or parentheses. Thus, we might write $A [1]$, $A [2]$, $A [3]$, etc., for the coefficients of the polynomial and $B [1, 1]$, $B [1, 2]$, etc., for the elements of the matrix.

We can enter the array $A(1), A(2), A(3), \dots A(10)$ into a program very simply by the lines:

```
10   FOR I = 1 TO 10
20   READ A (I)
30   NEXT I
40   DATA 2, 3, -5, 5, 2, 2, 4, -9, 123, 4, -4
```

We need no special instruction to the computer if no subscript greater than 10 occurs. However, if we want larger subscripts, we must use a DIM statement, to indicate to the computer that it has to save extra space for the array. When in doubt, indicate a larger dimension than you expect to use. The maximum value for a dimension is 255. For example, if we want a list of 15 numbers entered, we might write:

```
10   DIM A (25)
20   READ N
30   FOR I = 1 TO N
40   READ A (I)
50   NEXT I
60   DATA 15
70   DATA 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
```

Statements 20 and 60 could have been eliminated by writing 30 as $\text{FOR } I = 1 \text{ TO } 15$, but the form as typed would allow for the lengthening of the array by changing only statement 60, so long as it did not exceed 25.

We would enter a 3×5 array into a program by writing:

```
10   FOR I = 1 TO 3
20   FOR J = 1 TO 5
30   READ B (I, J)
40   NEXT J
50   NEXT I
60   DATA 2, 3, -5, -9, 2
```

```
70 DATA 4, -7, 3, 4, -2
80 DATA 3, -3, 5, 7, 8
```

Here again, we may enter an array with no dimension statement, and it will handle all the entries from B(1, 1) to B(10, 10). If you try to enter an array with a subscript greater than 10, without a DIM statement, you will get an error message telling you that you have a subscript error. This is easily rectified by entering the line:

```
5 DIM B (20, 30)
```

if, for instance, we need a 20-by-30 table.

The single letter denoting an array name may also be used to denote a simple variable without confusion. However, the same letter may not be used to denote both a singly subscripted and a doubly subscripted array in the same program. The form of the subscript is quite flexible, and you might have the array element B(I, K) or Q(A(3, 7), B - C).

Shown below is a list and run of a problem which uses both a singly and a doubly subscripted array. The program computes the total sales of each of five salesmen, all of whom sell the same three products. The array P gives the price/item of the three products and the array S tells how many items of each product each man sold. You can see from the program that product no. 1 sells for \$1.25 per item, no. 2 for \$4.30 per item, and no. 3 for \$2.50 per item; and also that salesman no. 1 sold 40 items of the first product, 10 of the second, and 35 of the third, and so on. The program reads in the price array in lines 40-80, using data in lines 910-930. The same program could be used again, modifying only line 900 if the prices change, and only lines 910-930 to enter the sales in another month.

This sample program did not need a dimension statement, since the computer automatically saves enough space to allow all subscripts to run from 1 to 10. A DIM statement is normally used to save more space. But in a long program, requiring many small arrays, DIM maybe used to save less space for arrays, in order to leave more for the program.

Since a DIM statement is not executed, it may be entered into the program on any line before END; it is convenient, however, to place DIM statements near the beginning of the program.

```
READY
10 FOR I = 1 TO 3
20 READ P(I)
30 NEXT I
40 FOR I = 1 TO 3
50 FOR J = 1 TO 5
60 READ S(I, J)
70 NEXT J
80 NEXT I
90 FOR J = 1 TO 5
100 LET S = 0
110 FOR I = 1 TO 3
120 LET S = S + P(I)*S(I, J)
130 NEXT I
140 PRINT "TOTAL SALES FOR SALESMAN "J, "$" S
```



```

150 NEXT J
900 DATA 1.25, 4.30, 2.50
910 DATA 40, 20, 37, 29, 42
920 DATA 10, 16, 3, 21, 8
930 DATA 35, 47, 29, 16, 33
999 END
RUN
TOTAL SALES FOR SALESMAN 1          $ 180.5
TOTAL SALES FOR SALESMAN 2          $ 211.3
TOTAL SALES FOR SALESMAN 3          $ 131.65
TOTAL SALES FOR SALESMAN 4          $ 166.55
TOTAL SALES FOR SALESMAN 5          $ 169.4

```

1.5 USE OF THE SYSTEM

Now that we know something about writing a program in BASIC, how do we set about using a teletypewriter to type in our program and then to have the computer solve our problem?

First, load the BASIC system tape and initiate its execution. The computer then types READY and you should begin to type your program. Make sure that each line begins with a line number which contains no more than four digits and contains no non-digit characters. Be sure to press the CARRIAGE RETURN key at the completion of each line. Spaces may be inserted at any point in the line, including before the line numbers.

If, in the process of typing a statement, you make a typing error and notice it immediately, you can correct it by pressing the backward arrow (shift key above the letter "oh"). This will delete the character in the preceding space, and you can then type in the correct character. Pressing this key a number of times will erase from this line the characters in that number of preceding spaces. To delete all of the present line, press ALT MODE.† Programs or data may be annotated by typing the remark and then deleting the line (as far as the system is concerned) with an ALT MODE. BASIC types a slash to show that a line has been deleted.

After typing your complete program, you type RUN, press the CARRIAGE RETURN key, and hope. If the program is one which the computer can run, it will then run it and type out any results for which you have asked in your PRINT statements. This does not mean that your program is correct, but that it has no errors of the type known as "grammatical errors". If it had errors of this type, the computer would have typed an error code as soon as the error was detected during the typing of the program. Errors detected after RUN are structural (loop nesting, matching GOSUB and Return) or arithmetical errors. A list of the error codes is in Sect. 2.7 together with the interpretation of each.

If you are given an error message, you can correct the error by typing a new line with the correct statement. If you want to eliminate the statement on line 110 from your program, you can do this by typing 110 and then the CARRIAGE RETURN. If you want to insert a statement between those on lines 60 and 70, you can do this by giving it a line number between 60 and 70.

† Use ESC key if no ALT MODE key.

If it is obvious to you that you are getting the wrong answers to your problem, even while the computer is running, you can type STOP and the computation will cease. If the teletypewriter is actually typing, there is an express stop - just press any key. It will type STOP and then READY and you can start to make your corrections. If you are in serious trouble, use the express stop, and type SCRATCH. When the system is ready to accept a new program, READY will be typed.

A sample use of the system is shown below.

```
READY
10 FOR N = 1 TO 7
20 PRINT N, SQR(N)
30 NEXT N
40 PRINT "DONE"
50 END
RUN
1          1
2          1.41421
3          1.73205
4          2
5          2.23607
6          2.44949
7          2.64575
DONE
```

1.6 ERRORS AND DEBUGGING

It may occasionally happen that the first run of a new problem will be free of errors and give the correct answers. But it is much more common that errors will be present and will have to be corrected. Errors are of two types: errors of form (or grammatical errors) which prevent the running of the program; and logical errors in the program which cause the computer to produce wrong answers or no answers at all.

Errors of form will cause error codes to be printed, and the various error codes are listed and explained in Sec. 2.7. Logical errors are often much harder to uncover, particularly when the program gives answers which seem to be nearly correct. In either case, after the errors are discovered, they can be corrected by changing lines, by inserting new lines, or by deleting lines from the program. As indicated in the last section, a line is changed by typing it correctly with the same line number; a line is inserted by typing it with a line number between those of two existing lines; and a line is deleted by typing its line number and pressing the CARRIAGE RETURN key. Notice that you can insert a line only if the original line numbers are not consecutive integers. For this reason, most programmers will start out using line numbers that are multiples of five or ten, but that is a matter of choice.

These corrections can be made at any time - whenever you notice them - either before or after a run. Since the computer sorts lines out and arranges them in order, a line may be retyped out of sequence. Simply retype the offending line with its original line number.

As with most problems in computing, we can best illustrate the process of finding the errors (or "bugs") in a program, and correcting (or "debugging") it, by an example. Let us consider the problem of finding that value of X between 0 and 3 for

which the sine of X is a maximum, and ask the machine to print out this value of X and the value of its sine. If you have studied trigonometry, you know that $\pi/2$ is the correct value; but we shall use the computer to test successive values of X from 0 to 3, first using intervals of .1, then of .01, and finally of .001. Thus, we shall ask the computer to find the sine of 0, of .1, of .2, of .3..., of 2.8, of 2.9, and of 3, and to determine which of these 31 values is the largest. It will do it by testing $\text{SIN}(0)$ and $\text{SIN}(.1)$ to see which is larger, and calling the larger of these two numbers M . Then it will pick the larger of M and $\text{SIN}(.2)$ and call it M . This number will be checked against $\text{SIN}(.3)$, and so on down the line. Each time a larger value of M is found, the value of X is "remembered" in $X0$. When it finishes, M will have been assigned to the largest value. It will then repeat the search, this time checking the 301 numbers 0, .01, .02, .03, ..., 2.98, 2.99, and 3, finding the sine of each and checking to see which has the largest sine. At the end of each of these three searches, we want the computer to print three numbers: the value $X0$ which has the largest sine, the sine of that number, and the interval of search.

Before going to the teletypewriter, we write a program and let us assume that it is the following:

```

10 READ D
20 LET X0 = 0
30 FOR X = 0 TO 3 STEP D
40 IF SIN (X) <= M THEN 100
50 LET X0 = X
60 LET M = SIN (X0)
70 PRINT X0, X, D
80 NEXT X0
90 GO TO 20
100 DATA .1, .01, .001
110 END

```

We shall list the entire sequence on the teletypewriter and make explanatory comments.

```

READY
1 REM PROGRAM NAME: MAXSIN
10 READ D
20 LWR X0= 0
ERROR 4 IN LINE 20

20 LET X0= 0
30 FOR X = 0 TO 3 STEP D
40 IF SINE-(X) <= M THEN 100
50 LET X0=X
60 LET M = SIN(X)
70 PRINT X0, X, D
ERROR 21 IN LINE 70

70 PRINT X0, X, D
80 NEXT Z=X0
90 GO TO 20
100 DATA .1, .01, .001

```

```
110 END
RUN
```

```
ERROR 41 IN LINE 80
```

A message indicates that LET was mistyped in line 20, so we retype it, this time correctly.

Notice the use of the backwards arrow to erase a character in line 40, which should have started IF SIN(X) etc., and in line 80.

After receiving the second error message, we find that we used XO for a variable instead of X0 in line 70. The next error message indicates a FOR statement without a NEXT. Upon checking we see that the variable in the FOR and NEXT are different, so we correct statement 80. In looking over the program, we also notice that the IF-THEN statement in 40 directed the computer to a DATA statement and not to line 80 where it should go.

```
80 NEXT X
40 IF SIN(X) <= M THEN 80
RUN
```

```
ERROR 50 IN LINE 40
```

The message indicates that M has never been assigned an initial value. We decide to give it a value less than the maximum value of the sine, say -1.

```
20 LET M= -1
RUN
0          0          .1
.1         .1         .1
.2         .2         .1
.3         .3         .1
.4         STOP
```

```
READY
```

This is incorrect. We are having every value of X0, X, and the interval size printed, so we direct the machine to cease operations by typing S even while it is running. Note that the 'S' does not print, but the word STOP is printed.

We fix this by moving the PRINT statement outside the loop. Typing 70 deleted that line, and line 85 is outside of the loop. We also realize that we want M printed and not X.

```
70
85 PRINT X0, M, D
RUN
1.6        .999574    .1
1.6        .999574    .1
1.6        .999574    .1
1.6        STOP
```

```
READY
```

Of course, line 90 sent us back to line 20 to repeat the operation and not back to line 10 to pick up a new value for D. We also decide to put in headings for our columns by a PRINT statement.

```
90 GO TO 10
5 PRINT "X VALUE", "SIN", RESOLUTION"
ERROR 21 IN LINE 5
```

There is an error in our PRINT statement: no left quotation mark for the third item

Retype line 5, with all of the required quotation marks.

```
5 PRINT "X VALUE", "SIN", RESOLUTION"
5 PRINT "X VALUE", "SIN", "RESOLUTION"
RUN
X VALUE          SIN          RESOLUTION
1.6              .999574      .1
1.57             1.          1.00000E-02
1.57098         1           1.00000E-03

ERROR 56 IN LINE 10
```

Exactly the desired results. Of the 31 numbers (0, .1, .2, .3, ..., 2.8, 2.9, 3) it is 1.6 which has the largest sine, namely .999574. Similarly for finer subdivisions.

Having changed so many parts of the program, we ask for a list of the corrected program. Listing the corrected program, from time to time, is an important part of debugging.

```
LIST
1 REM PROGRAM NAME: MAXSIN
5 PRINT "X VALUE", "SIN", "RESOLUTION"
10 READ D
20 LET M=-1
30 FOR X= 0 TO 3 STEP D
40 IF SIN(X) <= M THEN 80
50 LET X0=X
60 LET M=SIN(X)
80 NEXT X
85 PRINT X0,M,D
90 GOTO 10
100 DATA .1 , 1.00000E-02, 1.00000E-03
110 END

PLIST
```

The program is saved for later use by punching it on the Tape Punch.

In solving this problem, there is a common device which we did not use, namely the insertion of a PRINT statement when we wonder if the machine is computing what we think we asked it to compute. For example, if we wondered about M, we could have inserted 65 PRINT M, and we would have seen the values.

1.7 SUMMARY OF ELEMENTARY BASIC STATEMENTS

In this section we shall give a short and concise description of each of the types of BASIC statements discussed earlier in this chapter and add one statement to our list. In each form, we shall assume a line number, and shall use brackets to denote a general type. Thus, [variable] refers to any variable, which is a single letter, possibly followed by a single digit.

1.7.1 LET

This statement is not a statement of algebraic equality, but rather a command to the computer to perform certain computations and to assign the answer to a certain variable. Each LET statement is of the form: LET [variable] = [formula]. More generally several variables may be assigned the same value by a single LET statement.

Examples: (of the first type):

```
100 LET X = X + 1
```

```
259 LET W7 = (W-X4 + 3)*(Z - A/(A - B)) -17
```

(of the second type):

```
50 LET X = Y3 = A(3, 1) = 1
```

```
90 LET W = Z = 3*X -4* X + 2
```

1.7.2 READ and DATA

We use a READ statement to assign to the listed variables values obtained from a DATA statement. Neither statement is used without one of the other type. A READ statement causes the variables listed in it to be given, in order, the next available numbers in the collection of DATA statements. Before the program is run, the computer takes all of the DATA statements in the order in which they appear and creates a large data block. Each time a READ statement is encountered anywhere in the program, the data block supplies the next available number or numbers. If the data block runs out of data, with a READ statement still asking for more, the program is assumed to be done and we get an out-of-data error code.

Since we have to read in data before we can work with it, READ statements normally occur near the beginning of a program. The location of DATA statements is arbitrary, as long as they occur in the correct order. A common practice is to collect all DATA statements and place them just before the END statement.

Each READ statement is of the form:

READ { sequence of variables } and each DATA statement of the form:
DATA { sequence of numbers }

Examples: 150 READ X, Y, Z, X1, Y2, Q9
 330 DATA 4, 2, 1.7
 340 DATA 6.734E-3, -174.321, 3.14159265
 234 READ B (K)
 263 DATA 2, 3, 5, 7, 9, 11, 10, 8, 6, 4

```
10 READ R (I, J)
440 DATA -3, 5, -9, 2.37, 2.9876, -437.234E-5
450 DATA 2.765, 5.5576, 2.3789E2
```

Remember that only numbers are put in a DATA statement, and that $15/7$ and $\sqrt{3}$ are formulas, not numbers.

1.7.3 PRINT

The PRINT statement has a number of different uses and is discussed in more detail in Chapter 2. The common uses are (a) to print out the result of some computations, (b) to print out verbatim a message included in the program, (c) a combination of the two, and (d) to skip a line. We have seen examples of only the first two in our sample programs. Each type is slightly different in form, but all start with PRINT after the line number.

```
Examples of type (a): 100 PRINT X, SQR (X)

                     135 PRINT X, Y, Z, B*B -4*A*C, EXP(A-B)
```

The first will print X and then, a few spaces to the right of that number, its square root. The second will print five different numbers:

X, Y, Z, $B^2 - 4AC$, and e^{A-B} .

The computer will compute the two formulas and print them for you, as long as you have already given values to A, B, and C. It can print up to five numbers per line in this format.

```
Examples of type (b): 100 PRINT "NO UNIQUE SOLUTION"
                     430 PRINT "X VALUE", "SINE", "RESOLUTION"
```

Both have been encountered in the sample programs. The first prints that simple statement; the second prints the three labels with spaces between them. The labels in 430 automatically line up with three numbers called for in a PRINT statement (as long as the labels do not exceed 14 characters) as seen in MAXSIN.

```
Examples of type (c): 150 PRINT "THE VALUE OF X IS" X
                     30 PRINT "THE SQUARE ROOT OF "X. "IS"
                       SQR(X)
```

If the first has computed the value of X to be 3, it will print out: THE VALUE OF X is 3. If the second has computed the value of X to be 625, it will print out: THE SQUARE ROOT OF 625 IS 25.

```
Examples of type (d): 250 PRINT
```

The computer will advance the paper one line when it encounters this command.

1.7.4 GO TO

There are times in a program when you do not want all commands executed in the program. An example of this occurs in the MAXSIN problem where the computer has computed X0, M, and D and printed them out in line 85. We did not want the program to go to the END statement yet, but to go through the same process for a different value of D. So we directed the computer to go back to line 10 with a GO TO statement. Each is in the form of GO TO [line number]. (It is possible to go to a non-executable statement; control passes to the next sequential executable statement.)

Example: 150 GO TO 75

1.7.5 IF - THEN

There are times when we are interested in jumping the normal sequence of commands, if a certain relationship holds. For this we use an IF-THEN statement, sometimes called a conditional GO TO statement. Such a statement occurred at line 40 of MAXSIN. The more common form of the statement is:

IF [formula] [relation] [formula] THEN [line number]

Examples: 40 IF SIN (X) < = M THEN 80
 20 IF G = 0 THEN 65

The first asks if the sine of X is less than or equal to M, and directs the computer to skip to line 80 if it is. The second asks if G is equal to 0, and directs the computer to skip to line 65 if it is. In each case, if the answer to the question is No, the computer will go to the next line of the program.

In the general sense, the IF-THEN statement may have the form:

IF [formula] THEN [line number]

Examples: 17 IF G = 0 THEN 77
 35 IF A THEN 83
 85 IF A + B - 5 THEN 302
 90 IF -2 THEN 200

The formula is evaluated and the result interpreted as non-zero (true) or zero (false). Thus, in the first case, if G had a value of 2, the formula $G = 0$ is evaluated as zero or false. If A would have a value of 2 (in the second case), the formula is evaluated as non-zero or true. In the third case, if A equalled 2 and B equalled 3, the formula would yield a value of zero (false). The fourth example would always be true.

FOR [variable] = [formula] TO [formula] STEP [formula]

1.7.6 FOR and NEXT

We have already encountered the FOR and NEXT statements in our loops, and have seen that they go together, one at the entrance to the loop and one at the exit, directing the computer back to the entrance again. Every FOR statement is of the form

Any simple (not subscripted) variable may be used as the FOR variable. Most commonly, the expressions will be integers and the STEP omitted. In the latter case, a step size of one is assumed. The accompanying NEXT statement is simple in form, but the variable must be precisely the same one as that following FOR in the FOR statement. Its form is NEXT [variable].

Examples: 30 FOR X = 0 TO 3 STEP D
 80 NEXT X
 120 FOR X4 = (17 + COS(Z))/3 TO 3*SQR(10) STEP 1/4
 235 NEXT X4
 240 FOR X = 8 TO 3 STEP -1
 456 FOR J = -3 TO 12 STEP 2

Notice that the step size may be a formula (1/4), a negative number (-1), or a positive number (2). In the example with lines 120 and 235, the successive values of X4 will be .25 apart, in increasing order. In the next example, the successive values of X will be 8, 7, 6, 5, 4, 3. In the last example, on successive trips through the loop, J will take on values -3, -1, 3, 5, 7, 9, and 11.

If the initial, final, or step-size values are given as formulas, these formulas are evaluated once and for all upon entering the FOR statement. The control variable can be changed in the body of the loop; of course, the exit test always uses the latest value of this variable.

If you write 50 FOR Z = 2 TO -2, without a negative step size, the body of the loop will not be performed and the computer will proceed to the statement immediately following the corresponding NEXT statement.

1.7.7 DIM

Whenever we want to enter an array with a subscript greater than 10, we must use a DIM statement to inform the computer to save us sufficient room.

Examples: 20 DIM H(35)
 35 DIM Q(5, 25)

The first would enable us to enter an array of 35 items, and the latter a 5×25 array.

1.7.8 END

Every program must have an END statement, and it must be the statement with the highest line number in the program. Its form is simple: a line number with END.

Example: 999 END

2.1 MORE ABOUT PRINT

The uses of the PRINT statement were described in 1.7.3, but we shall give more detail here. Although the format of answers is automatically supplied for the beginner, the PRINT statement permits a greater flexibility for the more advanced programmer who wishes a different format for his output.

The teletypewriter line is divided into five zones starting at positions 0, 15, 30, 45, and 60. Control of the use of these comes from the use of the comma; a comma is a signal to move to the next print zone or, if the fifth print zone has just been filled, to move to the first print zone of the next line.

More compact output can be obtained by use of the semi-colon; it inhibits spacing between printzones on a line, acting only to separate quantities to be printed (e.g., A + B; C/D) or suppress a CARRIAGE RETURN at the end of a PRINT statement.

Spacing within a print zone depends on the value and type of the number being printed. A number is always printed in a zone larger than it needs and is left-justified in the zone. The size is determined as follows:

<u>Value of Number</u>	<u>Type of Number</u>	<u>Format of Zone</u>
$-999 \leq n \leq +999$	Integer	$\Delta xxx\wedge\wedge\wedge\uparrow$ only 6 long
$-32768 \leq n \leq -1000$ $+1000 \leq n \leq +32767$	Integer	$\Delta xxxxx\wedge\wedge\wedge$
$.1 \leq n \leq 999999.5$	Large Integer or Real	$\Delta xxxxxxx\wedge\wedge\wedge$ only 6 digits and decimal point. (Decimal point printed as one of x's; trailing zeros suppressed.)
$n < .1$ $999999.5 < n$	Large Integer or Real	$\Delta x.xxxxxE\pm ee\wedge\wedge\wedge$

For Example, if you were to type the program

```
10 FOR I = 1 TO 15
20 PRINT I
30 NEXT I
40 END
RUN
```

†The carot symbol, \wedge , represents a space typed on the teletypewriter.

the teletypewriter would print 1 at the beginning of a line, 2 at the beginning of the next line, and so on, finally printing 15 on the fifteenth line. But, by changing line 20 to read

```
20 PRINT I,  
RUN
```

you would have the numbers printed in the zones, reading

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

If you wanted the numbers printed in this fashion, but more tightly packed, you would change line 20 to replace the comma by a semi-colon:

```
20 PRINT I;  
RUN
```

and the result would be printed

1	2	3	4	5	6	7	8	9	10	11	12
13	14	15									

You should remember that a label inside quotation marks is printed just as it appears and also that the end of a PRINT signals a new line, unless a comma or semi-colon is the last symbol.

Thus, the instruction

```
50 PRINT X, Y
```

will result in the printing of two numbers and the return to the next line, while

```
50 PRINT X, Y,
```

will result in the printing of these two values and no return — the next number to be printed will occur in the third zone, after the values of X and Y in the first two.

Since the end of a PRINT statement signals a new line, you will remember that

```
250 PRINT
```

will cause the typewriter to advance the paper one line. It will put a blank line in line in your program, if you want to use it for vertical spacing of your results, or it causes the completion of partially filled line, as illustrated in the following fragment of a program:

```
50 FOR M = 1 TO N  
110 FOR J = 1 TO M  
120 PRINT B (M, J);  
130 NEXT J  
140 PRINT  
150 NEXT M
```

This program will print B(1, 1) and next to it B(1, 2). Without line 140, the teletypewriter would then go on printing B(2, 1), B(2, 2), and B(2, 3) on the same line, and then B(3, 1), B(3, 2) etc. Line 140 directs the teletypewriter, after printing the B(1, 2) value corresponding to M = 2, to start a new line and to do the same thing after printing the value of B(2, 3) corresponding to M = 3, etc.

The instructions

```
50 PRINT " HP BASIC ";  
51 PRINT "LANGUAGE COMPILER"  
90 END  
RUN
```

will result in the printing of

HP BASIC LANGUAGE COMPILER

Formatting of output can be controlled even further by use of the function TAB.

Insertion of TAB(17) will cause the teletypewriter to move to column 17, just as if a tab had been set there. For this purpose the positions on a line are numbered from 0 through 71.

More precisely, TAB may contain any formula as its argument. The value of the formula is computed, and its integer part is taken. (If the result is greater than 71, the teletypewriter is moved to position zero of the next line.) The teletypewriter is then moved forward to this position - unless it has already passed this position, in which case the TAB is ignored.

For example, inserting the following line in a loop:

```
PRINT X; TAB(12); Y; TAB(27); Z
```

will cause the X-value to start in column 0, the Y-value in column 12 and the Z-value in column 27.

A comma following a TAB clause has no effect on positioning the teletypewriter. For example, the statement "PRINT TAB (7), A+B" causes the value of A+B to be printed starting at position 7, while the statement "PRINT Z, A+B" causes the value to be printed starting at position 15.

The following rules for the printing of numbers will help you in interpreting your printed results:

1. If a number is an integer with a value between -32768 and +32767, inclusive, the decimal point is not printed.
2. If the number is an integer out of the above range (see INT function), or if the number is real and has an absolute value between .1 and 999999.5, the number is rounded to six digits and printed with a decimal point. Zeros trailing the decimal point are suppressed.

3. If a number is either greater than 999999.5 or less than .1 in magnitude, it is rounded to six places; the teletypewriter then prints a space (if positive) or minus sign (if negative), the first digit, the decimal point, the next five digits, the letter E (indicating exponent), the sign of the exponent, and the exponent. For example, it will take 32, 437, 580, 259 and write it as 3.24376E + 10.

The following program, in which we print out powers of 2, shows how numbers are printed.

```

READY
10 FOR N = -5 TO 30
20 PRINT 2*N;
30 NEXT N
40 END
RUN
3.12500E-02      6.25000E-02      .125      .25      .5      1
2      4      8      16      32      64      128      256      512      1024      2048
4096      8192      16384      32768.      65536.      131072.      262144.
524288.      1.04858E+06      2.09715E+06      4.19430E+06      8.38861E+06
1.67772E+07      3.35544E+07      6.71089E+07      1.34218E+08      2.68435E+08
5.36871E+08      1.07374E+09

```

2.2 FUNCTIONS AND DEF

Three functions were listed in Section 1.2 but not described: INT, RND and SGN.

2.2.1 INT

The INT function is the function which frequently appears in algebraic computation as $[x]$, and it gives the greatest integer not greater than x for $-32768 \leq x < 32768$. Thus $\text{INT}(2.35) = 2$, $\text{INT}(-2.35) = -3$, and $\text{INT}(12) = 12$. $\text{INT}(x) = 32767$ for $x \geq 32768$ and $\text{INT}(x) = -32768$ for $x \leq -32768$.

One use of the INT function is to round numbers. We may use it to round to the nearest integer by asking for $\text{INT}(X + .5)$. This will round 2.9, for example, to 3, by finding $\text{INT}(2.9 + .5) = \text{INT}(3.4) = 3$. You should convince yourself that this will indeed do the rounding guaranteed for it (it will round a number midway between two integers up to the larger of the integers).

It can also be used to round to any specific number of decimal places. For example, $\text{INT}(10*X + .5)/10$ rounds X correct to one decimal place, and $\text{INT}(10^D*X + .5)/10^D$ rounds X correct to D decimal places.

2.2.2 RND

The function RND produces a random number between 0 and 1. The form of RND requires an argument although the argument has no significance, and so we write $\text{RND}(X)$ or $\text{RND}(\emptyset)$. (The argument may be a constant or a previously defined variable.)

2-4 BASIC

If we want the first twenty random numbers, we write the program below and we get twenty six-digit decimals. This is illustrated in the following program.

```

READY
10 FOR L = 1 TO 20
20 PRINT RND(0),
30 NEXT L
40 END
RUN
1.52602E-05      .500092      .500412      1.64799E-03      6.17992E-03
.522248          .577867      .266971      .901025          .503415
.411261          .436832      .419642      8.63642E-02      .241408
.171168          .35434       8.55276E-02  .824103          .674869
READY
RUN
1.52602E-05      .500092      .500412      1.64799E-03      6.17992E-03
.522248

```

Note that the second RUN was giving exactly the same "random" numbers as the first RUN. This greatly facilitates the debugging of programs that use the random number generator.

On the other hand, if we want twenty random one-digit integers, we could change line 20 to read

```
20 PRINT INT(10*RND(0)),
```

and we would then obtain

0	5	5	0	0
5	5	2	9	5
4	4	4	0	2
1	3	0	8	6

We can vary the type of random numbers we want. For example, if we want 20 random numbers ranging from 1 to 9 inclusive, we could change line 20 as shown

```
20 PRINT INT(9*RND(0) + 1);
RUN
```

1	5	5	1	1	5	6	3	9	5	4	4
4	1	3	2	4	1	8	7				

or we can obtain random numbers which are integers from 5 to 24 inclusive by changing line 20 as in the following example.

```
20 PRINT INT(20*RND(0) + 5);
RUN
```

5	15	15	5	5	15	16	10	23	15	13	13
13	6	9	8	12	6	21	18				

In general, if we want our random numbers to be chosen from the A integers of which B is the smallest, we would call for $\text{INT}(A * \text{RND}(0) + B)$.

2.2.3 SGN

The SGN function is one which assigns the value 1 to any positive number, 0 to zero, and -1 to any negative number. Thus $\text{SGN}(7.23) = 1$, $\text{SGN}(0) = 0$, and $\text{SGN}(-.2387) = -1$.

2.2.4 DEF

In addition to the standard functions, you can define any other function which you expect to use a number of times in your program by use of a DEF statement. The name of the defined function must be three letters, the first two of which are FN. Hence, you may define up to 26 functions, e.g., FNA, FNB, etc.

The handiness of such a function can be seen in a program where you frequently need the function $e^{-x^2} + 5$. You would introduce the function by the line

```
30 DEF FNE(X) = EXP(-X * 2 + 5)
```

and later on call for various values of the function by $\text{FNE}(.1)$, $\text{FNE}(3.45)$, $\text{FNE}(A+2)$, etc. Such definition can be a great time-saver when you want values of some function for a number of different values of the variable.

The DEF statement may occur anywhere in the program, and the expression to the right of the equal sign may be any formula which can be fitted onto one line. It may include any combination of other functions, including ones defined by different DEF statements, and it can involve other variables besides the ones denoting the argument of the function. Assuming FNR is defined by

```
70 DEF FNR(X) = SQR(2 + LOG(X) - EXP(Y*Z)*(X + SIN(2*Z)))
```

and you have previously assigned values to Y and Z, you can ask for $\text{FNR}(2.7)$. You can give new values to Y and Z before the next use of FNR.

The use of DEF is limited to those functions whose value may be computed within a single BASIC statement. However, there are often much more complicated functions or portions of a program which must be calculated at several points within the program. For these operations, the GOSUB statement may be used.

2.3 GOSUB AND RETURN

When a particular part of a program is to be performed more than one time, or possibly at several different places in the overall program, it is most efficiently programmed as a subroutine. The subroutine is entered with a GOSUB statement, where the number is the line number of the first statement in the subroutine. For example,

```
90 GOSUB 210
```

directs the computer to jump to line 210, the first line of the subroutine. The logical end of the subroutine should be a return command directing the computer to return to the earlier part of the program. For example,

```
350 RETURN
```

will tell the computer to go back to the first line numbered greater than 90, and to continue the program there.

2-6 BASIC

The following example, a program for determining the greatest common divisor of three integers using the Euclidean Algorithm, illustrates the use of a subroutine. The first two numbers are selected in lines 30 and 40 and their GCD is determined in the subroutine, lines 200-310. The GCD just found is called X in line 60, the third number is called Y in line 70, and the subroutine is entered from line 80 to find the GCD of these two numbers. This number is, of course, the greatest common divisor of the three given numbers and is printed out with them in line 90.

You may use a GOSUB inside a subroutine to perform yet another subroutine. This would be called "nested GOSUBS". GOSUBS may be nested nine deep. In any case, it is absolutely necessary that a subroutine be left only with a RETURN statement, using a GOTO or an IF-THEN to get out if a subroutine will not work properly. You may have several RETURNS in the subroutine so long as exactly one of them will be used.

```

READY
10 PRINT " A", " B", " C", "GCD"
20 READ A, B, C
30 LET X = A
40 LET Y = B
50 GOSUB 200
60 LET X = G
70 LET Y = C
80 GOSUB 200
90 PRINT A, B, C, G
100 GO TO 20
110 DATA 60, 90, 120
120 DATA 38456, 64872, 98765
130 DATA 32, 384, 72
200 LET Q = INT(X/Y)
210 LET R = X - Q*Y
220 IF R = 0 THEN 300
230 LET X = Y
240 LET Y = R
250 GO TO 200
300 LET G = Y
310 RETURN
320 END
RUN

```

A	B	C	GCD
60	90	120	30
38456.	64872.	98765.	1
32	384	72	8

ERROR 56 IN LINE 20

2.4 INPUT

There are times when it is desirable to have data entered during the running of a program. This is particularly true when one person writes the program and enters it into the machine's memory, and other persons are to supply the data. This may be done by an INPUT statement, which acts as a READ statement but does not draw numbers from a DATA statement. If, for example, you want the user to supply values for X and Y into a program, you will type

40 INPUT X, Y

before the first statement which is to use either of these numbers. When it encounters this statement, the computer will type a question mark and wait. The user types two numbers, separated by a comma, presses the CARRIAGE RETURN key and the computer goes on with the rest of the program.

Frequently an INPUT statement is combined with a PRINT statement to make sure that the user knows what the question mark is asking for. You might type

```
20 PRINT "YOUR VALUES OF X, Y, AND Z ARE";  
30 INPUT X, Y, Z
```

and the machine will type out

```
YOUR VALUES OF X, Y, AND Z ARE?
```

Without the semicolon at the end of line 20, the question mark would have been printed on the next line.

Data entered via an INPUT statement is not saved with the program. Furthermore, it may take a long time to enter a large amount of data using INPUT. Therefore, INPUT should be used only when small amounts of data are to be entered, or when it is necessary to enter data during the running of the program such as with game-playing programs.

2.5 SOME MISCELLANEOUS STATEMENTS

Several other BASIC statements that may be useful from time to time are STOP, REM and RESTORE. The COM statement, a special feature of HP BASIC, permits transfer of data between programs.

2.5.1 STOP

STOP is entirely equivalent to GO TO xxxx, where xxxx is the line number of the END statement in the program. It is useful in programs having more than one natural finishing point. For example, the following two program portions are exactly equivalent.

250 GO TO 999	250 STOP
.....
340 GO TO 999	340 STOP
.....
999 END	999 END

2.5.2 REM

REM provides a means for inserting explanatory remarks in a program. The computer completely ignores the remainder of that line, allowing the programmer to follow the REM with directions for using the program, with identifications of the parts of a long program, or with anything else that he wants. Although what follows REM is ignored, its line number may be used in a GO TO or IF-THEN statement.

2-8 BASIC

```

100 REM INSERT DATA IN LINES 900-998.  THE FIRST
110 REM NUMBER IS N, THE NUMBER OF POINTS.  THEN
120 REM THE DATA POINTS THEMSELVES ARE ENTERED, BY

200 REM THIS IS A SUBROUTINE FOR SOLVING EQUATIONS

.....

300 RETURN

.....

520 GOSUB 200

```

2.5.3 RESTORE

Sometimes it is necessary to use the data in a program more than once. The RESTORE statement permits reading the data as many additional times as it is used. Whenever RESTORE is encountered in a program, the computer restores the data block pointer to the first number. A subsequent READ statement will then start reading the data all over again. A word of warning - if the desired data are preceded by code numbers or parameters, superfluous READ statements should be used to pass over these numbers. As an example, the following program portion reads the data, restores the data block to its original state, and reads the data again. Note the use of line 570 to "pass over" the value of N, which is already known.

```

100 READ N
110 FOR I = 1 TO N
120 READ X

.....

200 NEXT I

.....

560 RESTORE
570 READ X
580 FOR I = 1 TO N
590 READ X

.....

```

2.5.4 COM

The COM statement allows one program to store information in memory for retrieval by a subsequent program (only one BASIC program can exist in the system at a given time). The form of the statement is the same as for the DIM statement with the word COM replacing DIM, thus the common area information is accessible only as an array. Care must be taken not to dimension the same array in both a COM and DIM statement. The COM statement must be the first entered and lowest numbered statement of the program.

The common area is a block of contiguous storage elements; two computer words per element. The elements are allotted in the order that the arrays appear in the COM statement; the elements within an array are stored row by row. It is the programmer's responsibility to see that the portions of the common area are accessed properly. Assuming that one program starts with the statement "1 COM A(10), B(3,3)" and another with "1 COM C(5), D(5), F(3,3)", the common area storage elements would be assigned as follows:

<u>Element Position</u>	<u>First Program Reference</u>	<u>Second Program Reference</u>
1	A(1)	C(1)
2	A(2)	C(2)
3	A(3)	C(3)
4	A(4)	C(4)
5	A(5)	C(5)
6	A(6)	D(1)
7	A(7)	D(2)
8	A(8)	D(3)
9	A(9)	D(4)
10	A(10)	D(5)
11	B(1, 1)	F(1, 1)
12	B(1, 2)	F(1, 2)
13	B(1, 3)	F(1, 3)
14	B(2, 1)	F(2, 1)
15	B(2, 2)	F(2, 2)
16	B(2, 3)	F(2, 3)
17	B(3, 1)	F(3, 1)
18	B(3, 2)	F(3, 2)
19	B(3, 3)	F(3, 3)

A reference in the first program to B(1, 1) would access the same element as a reference to F(1, 1) in the second program. If A contained only 9 elements, however, the B(1, 1) and F(1, 1) references would access different elements.

The length of the common area may vary between programs, but for any two programs, information may be transferred only via the portion which is common to both.

If the first program declares "1 COM A(10), B(5, 5)" and the succeeding program contains "1 COM D(10), E(5, 5), F(10)"; the values of F would be unpredictable. If the second program contained "1 COM D(10)" only, the contents of B would be destroyed.

2.6 MATRICES

A common and convenient interpretation of doubly subscripted arrays is as matrices. Although you can work out for yourself programs which involve matrix computations, there is a special set of twelve instructions for such computations. They are identified by the fact that each instruction must start with the word 'MAT'. They are

MAT READ A, B, C	Read the three matrices, their dimensions having been previously specified. Data is stored in the matrix row by row.
MAT C = ZER	Fill out C with zeros.
MAT C = CON	Fill out C with ones.
MAT C = IDN	Set up C as an identity matrix.
MAT PRINT A, B; C	Print the three matrices, with A and C in the regular format, but B closely packed.
MAT B = A	Set the matrix B equal to the matrix A.
MAT C = A + B	Add the two matrices A and B.
MAT C = A - B	Subtract the matrix B from the matrix A.
MAT C = A*B	Multiply the matrix A by the matrix B.
MAT C = TRN(A)	Transpose the matrix A.
MAT C = (K)*A	Multiply the matrix A by K. K, which must be in parentheses, may be a formula.
MAT C = INV(A)	Invert the matrix A.

These twelve statements, with the addition of DIM, make matrix computations easier, and in combination with the ordinary BASIC instructions make the language much more powerful. However, the user has to be careful to keep (and to understand) the conventions "built into" the language. We will discuss, below, the individual MAT statements.

The following convention has been adopted for MAT: If in a MAT instruction we have a matrix of dimension M-by-N, the rows are numbered 1, 2, ..., M, and the columns 1, 2, ..., N.

The DIM statement may simply indicate what the maximum dimension is to be. Thus, if we write

DIM M(20, 35)

then M may have up to 20 rows and up to 35 columns. This statement is to save enough space for the matrix, and hence, the only care at this point is that the dimensions declared are large enough to accommodate the matrix. (The assumed dimensions of 10 rows by 10 columns do not apply for a matrix involved in matrix computations.)

The actual dimensions of a matrix are established either when it is set up (by a DIM statement) or when they are defined in one of four MAT statements: MAT READ, MAT - ZER, MAT - CON, or MAT - IDN.

Thus,

10 DIM M(20, 7)

50 MAT READ M

will read a 20-by-7 matrix for M, while

50 MAT READ M(17, 30)

will read 17-by-30 matrix for M, provided sufficient space has been saved for it by writing, for example,

10 DIM M(20, 35).

The elements of a matrix are stored by row in ascending locations in memory; two computer words are used for each element. A matrix declared as DIM A(3, 3) will be structured as:

		Columns		
Rows	A(1, 1)	A(1, 2)	A(1, 3)	
	A(2, 1)	A(2, 2)	A(2, 3)	
	A(3, 1)	A(3, 2)	A(3, 3)	

The elements would be stored in the following order:

<u>Element Position</u>	<u>Memory Location</u>	<u>Element</u>
1	m	A(1, 1)
2	m+2	A(1, 2)
3	m+4	A(1, 3)
4	m+6	A(2, 1)
5	m+8	A(2, 2)
6	m+10	A(2, 3)
7	m+12	A(3, 1)
8	m+14	A(3, 2)
9	m+16	A(3, 3)

Given a statement DIM A(M, N), the location of element A(i, j) with respect to the first element, A(1, 1), of the matrix is given by the equation:

$$\text{location } A(1, 1) + 2(M(i-1) + (j-1))$$

The three instructions:

MAT M = ZER

MAT M = CON

MAT M = IDN

which set up a matrix M with all components zero, all components equal to one, and as an identity matrix, respectively, act like MAT READ as far as the dimension of the resulting matrix is concerned. For example,

MAT M = CON(7, 3)

sets up a 7-by-3 matrix with 1 in every component, while

MAT M = CON

sets up a matrix, with ones in every component, according to previously specified dimensions. Thus,

```
10 DIM M(20, 7)
20 MAT READ M(7, 3)
.....
35 MAT M = CON
.....
70 MAT M = ZER(15, 7)
.....
90 MAT M = ZER(16, 10)
```

will first read in a 7-by-3 matrix for M. Then it will set up a 7-by-3 matrix of all ones for M (the actual dimension having been set up as 7-by-3 in line 20.) Next it will set up M as a 15-by-7 all zero matrix. (Note that although this is larger than the previous M, it is within the limits set in 10.) But it will result in an error code in line 90. The limit set in line 10 is 140 components, and in 90 we are calling for 160 components. The original dimensions may be exceeded, providing the total number of components is within the proper limit;

```
90 MAT M = ZER(25, 5)
```

would not yield an error message.

The **MAT PRINT** statement causes the array elements to be printed row by row across the page. The spacing between row elements is controlled by the use of , and ; in the same manner as for the **PRINT** statement. Rows containing more elements than can be printed on a line are continued on consecutive lines. Each row is started on a new line and is separated from the previous row by a blank line. Thus the instruction

MAT PRINT A, B; C

will cause the three matrices to be printed A and C with five components to a line and B with up to twelve.

Singly subscripted arrays may be interpreted as column nestors. Vectors may be used in place of matrices, as long as the above rules are observed. Since a vector like **V(I)** is treated as a column vector by BASIC, a row vector has to be introduced as a matrix that has only one row, namely row 1. Thus

```
DIM X(7), Y (1, 5)
```

introduces a 7-component column vector and a 5-component row vector.

A column vector will be printed one element to the line with double spacing between lines. A row vector will be printed in the manner indicated by the form of the statement. For example: if V is a row vector then, "MAT PRINT V" will print the vector V a single element to the line; "MAT PRINT V," will print V as a row vector, five numbers to the line, while

MAT PRINT V;

will print V as a row vector with up to twelve numbers to the line

MAT B = A

This sets B up to be the same as A provided the dimensions previously assigned to B are the same as those of A.

MAT C = A + B, MAT C = A - B

For these to be legal A, B, and C must have the same dimensions. Instructions MAT A = A + B are legal - the indicated operation is performed and the answer stored in A. Only a single arithmetic operation is allowed so MAT D = A + B - C is illegal but may be achieved with two MAT instructions.

MAT C = A * B

For this to be legal it is necessary that the number of columns in A be equal to the number of rows in B, the number of rows in C be equal to the number of rows in A, and the number of columns in C be equal to the number of columns in B. For example, if A has dimension L-by-M and B has dimension M-by-N then C = A * B must have dimension L-by-N. It should be noted that while MAT A = A + B may be legal, MAT A = A * B will result in nonsense. There is good reason for this. In adding two matrices we may immediately store the answer in one of the matrices; but if we attempt to do this in multiplying two matrices, we will destroy components which would be needed to complete the computation. MAT B = A * A is, of course, legal provided A is a 'square' matrix.

MAT C = TRN(A)

This lets C be the transpose of the matrix A. If A is an M-by-N matrix C must be an N-by-M matrix.

MAT C = (K) * A

The matrix C is the matrix A multiplied by the value of K (i.e., each component of A is multiplied by K to form the components of C). K, which must be in parentheses, may be a number or a formula. MAT A = (K) * A is legal.

MAT C = INV(A)

The number C is the inverse of A. (A must, of course, be a 'square' matrix.) A matrix must not be inverted or transposed into itself.

We close this section with two illustrations of matrix programs. The first one reads in A and B in line 30 and in so doing sets up the correct dimensions. Dimensions

for C are set up in line 35. Then, in line 40, $A + A$ is computed and the answer is called C. Note that the data in line 90 results in A being 2-by-3 and B being 3-by-3. Both MAT PRINT formats are illustrated, and one method of labeling a matrix print is shown.

```

READY
10 DIM A(15,15), B(15,15), C(15,15)
20 READ M,N
30 MAT READ A(M,N), B(N,N)
35 MAT C = ZER(M,N)
40 MAT C = A + A
50 MAT PRINT C;
60 MAT C = A*B
70 PRINT
75 PRINT "A*B ="
80 MAT PRINT C,
90 DATA 2, 3
91 DATA 1, 2, 3
92 DATA 4, 5, 6
93 DATA 1, 0, -1
94 DATA 0, -1, -1
95 DATA -1, 0, 0
99 END
RUN
  2      4      6

  8      10     12

```

```

A*B =
-2      -2      -3

-2      -5      -9

```

The second example inverts an N-by-N Hilbert Matrix

1	1/2	1/3	.	.	1/N
1/2	1/3	1/4	.	.	1/N+1
1/3	1/4	1/5	.	.	1/N+2
.
.
.
1/N	1/N+1	1/N+2	.	.	1/2N-1

Ordinary BASIC instructions are used to set up the matrix in lines 50 to 90. Note that this occurs after correct dimensions have been declared. Then a single instruction results in the computation of the inverse, and one more instruction prints it. In this example we have supplied 4 for N in the DATA statement and have made a run for this case.


```

READY
5  REM THIS PROGRAM INVERTS AN N-BY-N MATRIX
10 DIM A(20,20), B(20,20)
20 READ N
30 MAT A = CON(N,N)
40 MAT B = CON(N,N)
50 FOR I = 1 TO N
60 FOR J = 1 TO N
70 LET A(I,J) = 1/(I+J-1)
80 NEXT J
90 NEXT I
100 MAT B = INV(A)
110 PRINT
115 PRINT "INV(A) ="
120 PRINT
125 MAT PRINT B;
190 DATA 4
199 END
RUN

```

INV(A) =

16.0019	-120.021	240.049	-140.031
-120.02	1200.22	-2700.52	1680.33
240.048	-2700.51	6481.2	-4200.77
-140.03	1680.33	-4200.77	2800.49

The reader is warned that beyond $N = 7$ the Hilbert matrix cannot be inverted because of severe round-off errors.

2.7 ERROR CODES

An error message is printed as soon as the error condition is detected. The format is as follows:

ERROR xx IN LINE nn

Error Code	Meaning
1	Statement ends unexpectedly.
2	Input exceeds 71 characters.
3	System command not recognized. (May be missing statement number.)
4	Missing or incorrect statement type.
5	Exponent of number is missing power.
6	Symbol following MAT not recognized.
7	LET statement has no store.
8	Multiple or misplaced COM statement.

- 9 Missing or incorrect function identifier in DEF.
- 10 Missing parameter in DEF statement.
- 11 Missing assignment operator.
- 12 Missing THEN.
- 13 Missing or incorrect for-variable.
- 14 Missing TO.
- 15 Incorrect STEP in FOR statement.
- 16 Called routine does not exist.
- 17 Wrong number of parameters in CALL statement.
- 18 Missing or incorrect constant in DATA statement.
- 19 Missing or incorrect variable in READ statement.
- 20 No closing quote for PRINT string.
- 21 Missing print delimiter or bad PRINT quantity.
- 22 Illegal word follows MAT.
- 23 Missing delimiter.
- 24 Improper matrix function.
- 25 No subscript where expected.
- 26 May not invert or transpose matrix into self.
- 27 Missing multiplication operator.
- 28 Improper matrix operator.
- 29 Matrix may not be both operand and result of matrix multiplication.
- 30 Missing left parenthesis.
- 31 Missing right parenthesis.
- 32 Operand not recognized.
- 33 Defined array missing subscript part.
- 34 Missing array identifier.
- 35 Missing or bad integer.
- 36 Non -blank characters following statement's logical end.
- 37 Out of storage during syntax phase.
- 38 Punched Tape Reader not ready.
- 39 Doubly defined function.
- 40 FOR statement has no matching NEXT statement.
- 41 NEXT statement has no matching FOR statement.
- 42 Out of storage for symbol table.
- 43 Array appears with inconsistent dimensions.

- 44 Last statement is not END.
- 45 Array doubly dimensioned.
- 46 Number of dimensions not obvious.
- 47 Array too large.
- 48 Out of storage during array allocation.
- 49 Subscript exceeds bound.
- 50 Accessed operand has undefined value.
- 51 Non-integer power of negative number.
- 52 Zero to zero power.
- 53 Missing statement.
- 54 Gosubs nested 10 deep.
- 55 RETURN finds no address.
- 56 Out of data.
- 57 Out of storage during execution.
- 58 Dynamic array exceeds allocated storage.
- 59 Dimensions not compatible.
- 60 Matrix operand contains undefined element.
- 61 Singular or nearly singular matrix.
- 62 Trigonometric function argument is too large.
- 63 Attempted square root of negative argument.
- 64 Attempted log of negative argument.

The following errors are warning only, execution continues.

- 65 Numerical overflow, result taken to be + or -infinity.
- 66 Numerical underflow, result taken to be zero.
- 67 Log of zero taken to be -infinity.
- 68 EXP overflows, result taken to be +infinity.
- 69 Division by zero, result taken to be + or -infinity.
- 70 Zero raised to negative power, result taken to be +infinity.

To use the BASIC system, you load the tape into the computer, start execution of the system, and begin typing your program when the system types "READY." If desired, the program may also be entered from the Punched Tape Reader.

OPERATING INSTRUCTIONS

Load BASIC using the Basic Binary Loader.

1. Place BASIC binary tape in the Standard Input Unit (e. g. , Punched Tape Reader).
2. Set Switch Register to starting address of Basic Binary Loader: 017700
3. Press LOAD ADDRESS.
4. Set Loader switch to ENABLED. (On 2114A: LOADER ENABLE to ON.)
5. Press PRESET.
6. Press RUN.
7. When the computer halts with T-Register containing 102077, the BASIC tape is loaded. Set Loader switch to PROTECTED. (2114A: LOADER ENABLE to NORMAL.)

Set Switch Register to starting address of BASIC:

000100

Press LOAD ADDRESS and Run.

When the system types READY, begin typing program. Terminate each line with a CARRIAGE RETURN. (When the system is ready to accept a new line, it issues a LINE FEED.)

If errors are made while typing the program, they can be corrected by one of the following:

1. To delete one or more characters that have just been typed, press ← for each character, then type in the correct characters.
2. To delete the current line, press ALT MODE or ESC, and type the correct line.
3. To correct a previously typed line, retype the entire line starting with the line number.
4. To delete a previously typed line, type the line number followed by a CARRIAGE RETURN.

CONTROL COMMANDS

There are several commands that may be given to the computer by typing the command at the start of a new line (no line number) and following the command with a CARRIAGE RETURN.

STOP	Stops all operations at once. When the teletypewriter is typing, depressing any key will execute the STOP command. When the system is ready to accept further input, it types "READY".
RUN	Begins the computation of a program.
SCRATCH	Destroys the problem currently being worked on; it gives the user a "clean sheet" to work on. When the system is ready to accept a new program, it types "READY."
LIST	Causes an up-to-date listing of the program to be typed out. †
LIST XXXX	Causes an up-to-date listing of the program to be typed out beginning at line number XXXX and continuing to the end. †
TAPE	Informs the system that the program will be entered through the teletype reader. As the program tape will have line feeds following carriage returns, the feedback of a line feed after each statement will be suppressed to avoid double spacing between lines of the program.
PLIST	Causes an up-to-date copy of the program with leader, trailer to be punched on the High Speed Tape Punch. This tape may be saved and reloaded to execute the program at a later date. ①
PTAPE	Causes the system to read in the program from the Punched Tape Photoreader. ②

① If this command is given to a system which does not have a High Speed Tape Punch, the program is listed on the teletype preceded and followed by blank characters. Therefore to obtain a listing for debugging purposes type LIST, to obtain a copy of the program on paper tape for future use turn on the punch device appropriate to the system and type PLIST.

② If this command is given to a system which does not have a Photoreader, the system will type **STOP** and await further input from the teletype. **READY**

† The last few characters of statements whose listing exceeds 72 characters may not show on the listing but they are not lost internally.

HP BASIC provides two additional statements for use in a program.

WAIT

The WAIT statement extends BASIC to allow the introduction of delays into a program. Execution of WAIT (formula) causes the program to wait for the number of milliseconds, up to 32767, specified by the value of the formula. This is not precise because it does not take into account the time required to evaluate the formula.

CALL

The CALL statement is a provision of HP BASIC for interfacing absolute assembly language subroutines, typically input-output drivers, which can be added to the standard system to create specialized configurations. When executing a CALL statement, BASIC transfers control to an absolute assembly language subroutine which has been appended to the standard system. Once this transfer has taken place, the subroutine is in complete control of the computer and is at liberty to alter any part of core including the system itself. To minimize the danger of destroying the system by such an alteration, only those programmers who are proficient in absolute assembly language programming should attempt to append CALL statement subroutines to the standard system.

Subroutines which have been appended to BASIC can be accessed in a BASIC program through a statement of the form:

CALL (<subroutine number>, <parameter list>).

The subroutine number is a positive integer specifying the desired subroutine, if no such numbered subroutine exists the statement will be rejected by the syntax analyzer. The parameters, separated by commas, may be any formulas and their number is dependent upon the subroutine called. As an example, suppose that a subroutine designated by 5 has been appended to the system. Its function is to take readings from an A to D subsystem and store them in an array. The parameters specify the array into which the values are to be inserted, the channel number of the first point to be measured, the setup for the A to D converter and the number of points to be measured. A representative call might be as follows:

20 CALL (5, A[1], 1, 1188, 10)

 ↑ ↑ ↑ ↑ ↑
 Subroutine number First element of data array Starting channel number A to D setup Number of points

In using such statements, it is very important that correct parameters be supplied. Interchanging the first and second parameters could result in the destruction of the core-resident BASIC system, unless special precautions have been taken by the writer of the called subroutine.

PREPARATION OF SPECIAL SYSTEMS

Since subroutines accessed by CALL statements are a part of the BASIC system rather than a part of the user's BASIC program, they may be included on the composite tape produced by PBS. This can be accomplished by following the instructions given in Appendix E except that the absolute assembly binary tape of the subroutines is loaded between steps 2. and 3. The binary tape from step 8. replaces the normal BASIC system tape.

BASIC accesses called subroutines through a table containing linkage information. Entries in the table, one per subroutine, are two words in length. Bits 5-0 of the first word contain the number identifying the subroutine (chosen freely from 1 to 77₈ inclusive) and bits 15-8 contain the number of parameters passed to the subroutine (CALL statements with an incorrect number of parameters will be rejected by the syntax analyzer). The second word contains the absolute address of the entry point of the subroutine (control is transferred via a JSB). Although subroutine numbers need not be assigned in any particular order, all entries in the table must be contiguous. An acceptable auxiliary tape contains the following:

- 1) An ORG statement to origin the program at an address greater than that of the last word of the BASIC system. The address of this last word + 1 is contained in location 110₈ of the standard BASIC system. Hence, a suitable lower limit for the origin address can be determined by loading BASIC and examining location 110₈.
- 2) The subroutine linkage table described above.
- 3) The assembly language subroutines.
- 4) Code to set the following linkage addresses:
 - a) In location 110₈ put the address of the last word + 1 used in the auxiliary tape.
 - b) In location 121₈ put the address of the first word of the subroutine linkage table.
 - c) In location 122₈ put the address of the last word + 1 of the subroutine linkage table.

Assume that location 110₈ of the standard BASIC system contains 13405₈. An acceptable auxiliary tape could be assembled from the following code:

```
          ORG 13405B
SBTBL    OCT 2406    Subroutine 6 has 5 parameters
          DEF SB6
          OCT 1421    Subroutine 17 has 3 parameters
          DEF SB17
```


Transmittal of quantities of data between a BASIC program and a called subroutine is most conveniently handled through arrays. Since only addresses are passed to a subroutine, an array parameter must be an element of the array (in general this would be the first element of the array). When using arrays in this manner, it is important to remember that arrays are stored by rows, and that each element is a floating point number occupying two words. Hence, if an array A has M columns per row, the address of $A[I, J]$ is $(\text{address } A[1, 1] + 2(M(I-1) + (J-1)))$.

It may be necessary or desirable for a called subroutine to output on the teletype. This may be done by loading a buffer address into the B-register, a character count into the A-register, and executing a JSB 102B, I. The referenced block of core will be interpreted as an ASCII string and output accordingly, followed by a carriage return and line feed.

SYSTEM PROTECTION

Whenever data is transferred from a called subroutine through the address of a parameter, there is a danger that the BASIC system or users program might be destroyed. This situation can arise when parameters are specified incorrectly or insufficient space is allocated in a data array. For example, constants (e.g. 2 or -1.1) in a BASIC program are stored in the program as they appear, therefore storing through the address of a constant parameter will change the actual constant in the CALL statement. A subsequent execution of that statement may lead to strange results. A parameter that is an expression (e.g. A & B or NOT A AND B) will be evaluated and the result placed in a temporary location. Since the parameter address references this temporary, storing into it will not harm the BASIC system or program. However, the value stored there is lost to the BASIC program. If a called subroutine stores more values in an array than the array can hold, the resulting overflow of data may destroy the BASIC system or program. To provide some protection against these dangers, users of CALL statements should be cautioned against the danger of using unsuitable parameters in CALL statements (especially against using a simple variable or a constant where an array element is expected). Also, when using arrays as parameters it is good practice to include the dimensions of the array as additional parameters to allow a means of checking within the subroutine.

A really effective measure of protection is available at the cost of additional programming effort. BASIC contains sets of pointers delimiting the areas of memory within which different types of parameters exist. By checking parameter addresses against these bounds, the subroutine can verify that they are of the expected type. If X represents the parameter address, the following applies:

- a) Constant parameter (112B) <X <(113B)
- b) Simple variable parameter (116B) <X <(117B)
- c) Array parameter 1) In common storage (110B) <X <(112B)
2) Not in common storage (113B) <X <(115B)
- d) Expression parameter (115B) <X <(1120B)

where (112B) etc. means the contents of location number octal 112.

HP BASIC extends the concept of the formula to include logical operators. Included are the familiar relational operators =, # (not equal), <, >, <= (less than or equal), and >= (greater than or equal) and the Boolean operators AND, OR, and NOT. With the exception of NOT, which takes the following operand as its single argument, all of these are binary operators. In a formula the binary operators have a lower priority than any of the arithmetic operators †, *, /, +, and -. Among the binary operators the priority in ascending order is OR, AND, and the relational operators (all of equal priority). NOT has the same priority as the unary + and unary -. Thus

$$\text{NOT } A + B = C \text{ OR } \text{SGN}(K) \text{ AND } \text{SGN}(J)$$

is equivalent to

$$(((\text{NOT } A) + B) = C) \text{ OR } (\text{SGN}(K) \text{ AND } \text{SGN}(J)).$$

The relational operators take algebraic numbers as arguments and return 0 (false) or 1 (true) according to the relationship existing between their arguments. Thus $3 < 2$ evaluates to 0 and $A \neq 0$ evaluates to 1 if A has a non-zero value. The Boolean operators consider their arguments as zero (false) or non-zero (true) and return 0 or 1 as follows:

AND			OR			NOT	
Arg 1	Arg 2	Result	Arg 1	Arg 2	Result	Arg	Result
non-zero	non-zero	1	non-zero	non-zero	1	non-zero	0
non-zero	zero	0	non-zero	zero	1	zero	1
zero	non-zero	0	zero	non-zero	1		
zero	zero	0	zero	zero	0		

Thus 3 and 1 evaluates to 1 while $\text{NOT } -3 \text{ AND } 0$ evaluates to 0 . It is important to realize that $A < B < C$ is not equivalent to $A < B \text{ AND } B < C$. If $A = -3$, $B = -2$, and $C = 1/2$ the former evaluates as $(-3 < -2) < 1/2$ or 0 (false) whereas the latter evaluates as $(-3 < -2) \text{ AND } (-2 < 1/2)$ or 1 (true).

The syntax of HP BASIC is described in Backus Normal Form through the use of several metalinguistic symbols. Quantities within \langle and \rangle are metalinguistic variables representing a syntactic class. The symbol "::=" means "is defined as" and connects the metalinguistic variable on the left with its definition on the right. Multiple definitions of a metalinguistic variable are separated by the symbol "|", meaning "or". All capital letters and symbols not enclosed in \langle and \rangle are actual characters as they appear in the language (e.g., RESTORE or \langle integer \rangle . \langle integer \rangle). Uncapitalized letters represent English language expositions such as carriage return. Juxtaposition of quantities in a definition implies juxtaposition in the language; all BASIC punctuation appears explicitly.

Example: \langle integer $\rangle ::= \langle$ digit $\rangle | \langle$ integer $\rangle \langle$ digit \rangle explains that an \langle integer \rangle is either a \langle digit \rangle or an \langle integer \rangle immediately followed by a \langle digit \rangle . (Remember that blanks in BASIC only count within a character string, so that 1 23 is the same as 123.) It is easy to see that an integer begins with a digit and absorbs digits to its right one by one until it finds a non-digit character.

\langle basic program $\rangle ::= \langle$ program statement $\rangle | \langle$ basic program $\rangle \langle$ program statement \rangle ⁽¹⁾

\langle program statement $\rangle ::= \langle$ sequence number $\rangle \langle$ basic statement \rangle carriage return

\langle sequence number $\rangle ::= \langle$ integer \rangle ⁽²⁾

\langle basic statement $\rangle ::= \langle$ let statement $\rangle | \langle$ dim statement $\rangle | \langle$ com statement $\rangle |$
 \langle def statement $\rangle | \langle$ rem statement $\rangle | \langle$ go to statement $\rangle |$
 \langle if statement $\rangle | \langle$ for statement $\rangle | \langle$ next statement $\rangle | \langle$ gosub statement $\rangle |$
 \langle return statement $\rangle | \langle$ end statement $\rangle | \langle$ stop statement $\rangle |$
 \langle wait statement $\rangle | \langle$ call statement $\rangle | \langle$ data statement $\rangle |$
 \langle read statement $\rangle | \langle$ restore statement $\rangle | \langle$ input statement $\rangle |$
 \langle print statement $\rangle | \langle$ mat statement \rangle

\langle let statement $\rangle ::= \langle$ let head $\rangle \langle$ formula \rangle

\langle let head $\rangle ::= \text{LET } \langle$ variable $\rangle = | \langle$ let head $\rangle \langle$ variable $\rangle =$

\langle formula $\rangle ::= \langle$ conjunction $\rangle | \langle$ formula $\rangle \text{ OR } \langle$ conjunction \rangle

\langle conjunction $\rangle ::= \langle$ boolean primary $\rangle | \langle$ conjunction $\rangle \text{ AND } \langle$ boolean primary \rangle

\langle boolean primary $\rangle ::= \langle$ arithmetic expression $\rangle |$

\langle boolean primary $\rangle \langle$ relational operator $\rangle \langle$ arithmetic expression \rangle

$\langle \text{arithmetic expression} \rangle ::= \langle \text{term} \rangle | \langle \text{arithmetic expression} \rangle + \langle \text{term} \rangle |$
 $\langle \text{arithmetic expression} \rangle - \langle \text{term} \rangle$
 $\langle \text{term} \rangle ::= \langle \text{factor} \rangle | \langle \text{term} \rangle * \langle \text{factor} \rangle | \langle \text{term} \rangle / \langle \text{factor} \rangle$
 $\langle \text{factor} \rangle ::= \langle \text{primary} \rangle | \langle \text{sign} \rangle \langle \text{primary} \rangle | \text{NOT } \langle \text{primary} \rangle$
 $\langle \text{primary} \rangle ::= \langle \text{operand} \rangle | \langle \text{primary} \rangle \langle \text{operands} \rangle$
 $\langle \text{relational operator} \rangle ::= > | < | = | \neq$
 $\langle \text{operand} \rangle ::= \langle \text{variable} \rangle | \langle \text{unsigned number} \rangle | \langle \text{system function} \rangle |$
 $\langle \text{function} \rangle | \langle \text{formula operand} \rangle$
 $\langle \text{variable} \rangle ::= \langle \text{simple variable} \rangle | \langle \text{subscripted variable} \rangle$
 $\langle \text{simple variable} \rangle ::= \langle \text{letter} \rangle | \langle \text{letter} \rangle \langle \text{digit} \rangle$
 $\langle \text{subscripted variable} \rangle ::= \langle \text{array identifier} \rangle \langle \text{subscript head} \rangle$
 $\langle \text{subscript} \rangle \langle \text{right bracket} \rangle$
 $\langle \text{array identifier} \rangle ::= \langle \text{letter} \rangle$
 $\langle \text{subscript head} \rangle ::= \langle \text{left bracket} \rangle | \langle \text{left bracket} \rangle \langle \text{subscript} \rangle,$
 $\langle \text{subscript} \rangle ::= \langle \text{formula} \rangle$
 $\langle \text{letter} \rangle ::= \text{A} | \text{B} | \text{C} | \text{D} | \text{E} | \text{F} | \text{G} | \text{H} | \text{I} | \text{J} | \text{K} | \text{L} | \text{M} | \text{N} | \text{O} | \text{P} | \text{Q} | \text{R} | \text{S} | \text{T} | \text{U} | \text{V} | \text{W} | \text{X} | \text{Y} | \text{Z}$
 $\langle \text{digit} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$
 $\langle \text{left bracket} \rangle ::= (|$
 $\langle \text{right bracket} \rangle ::=) |$
 $\langle \text{sign} \rangle ::= + | -$
 $\langle \text{unsigned number} \rangle ::= \langle \text{decimal part} \rangle | \langle \text{decimal part} \rangle \langle \text{exponent} \rangle$
 $\langle \text{decimal part} \rangle ::= \langle \text{integer} \rangle | \langle \text{integer} \rangle . | \langle \text{integer} \rangle . \langle \text{integer} \rangle | . \langle \text{integer} \rangle$
 $\langle \text{integer} \rangle ::= \langle \text{digit} \rangle | \langle \text{integer} \rangle \langle \text{digit} \rangle$
 $\langle \text{exponent} \rangle ::= \text{E} \langle \text{integer} \rangle | \text{E} \langle \text{sign} \rangle \langle \text{integer} \rangle$
 $\langle \text{system function} \rangle ::= \langle \text{system function name} \rangle \langle \text{parameter part} \rangle$
 $\langle \text{system function name} \rangle ::= \text{SIN} | \text{COS} | \text{TAN} | \text{ATN} | \text{EXP} | \text{LOG} | \text{ABS} | \text{SQR} | \text{INT} | \text{RND} | \text{SGN}$
 $\langle \text{parameter part} \rangle ::= \langle \text{left bracket} \rangle \langle \text{actual parameter} \rangle \langle \text{right bracket} \rangle$
 $\langle \text{actual parameter} \rangle ::= \langle \text{formula} \rangle$
 $\langle \text{function} \rangle ::= \text{FN} \langle \text{letter} \rangle \langle \text{parameter part} \rangle$
 $\langle \text{formula operand} \rangle ::= \langle \text{left bracket} \rangle \langle \text{formula} \rangle \langle \text{right bracket} \rangle$
 $\langle \text{dim statement} \rangle ::= \text{DIM } \langle \text{formal array list} \rangle$
 $\langle \text{formal array list} \rangle ::= \langle \text{formal array} \rangle | \langle \text{formal array list} \rangle , \langle \text{formal array} \rangle$
 $\langle \text{formal array} \rangle ::= \langle \text{array identifier} \rangle \langle \text{formal bound head} \rangle \langle \text{formal bound} \rangle$
 $\langle \text{right bracket} \rangle$
 $\langle \text{formal bound head} \rangle ::= \langle \text{left bracket} \rangle | \langle \text{left bracket} \rangle \langle \text{formal bound} \rangle ,$


```

<string>::= "<character string>" (4)
<delimiter>::= , | ;
<print formula>::= <formula> | TAB <parameter part>
<mat statement>::= MAT <mat body>
<mat body>::= <mat read> | <mat print> | <mat replacement>
<mat read>::= READ <actual array> | <mat read>, <actual array>
<actual array>::= <array identifier> | <array identifier> <bound part>
<bound part>::= <actual bound head> <actual bound> <right bracket>
<actual bound head>::= <left bracket> | <left bracket> <actual bound>,
<actual bound>::= <formula>
<mat print>::= PRINT <mat print part> | PRINT <mat print part> <delimiter>
<mat print part>::= <array identifier> | <mat print part> <delimiter> <array identifier>
<mat replacement>::= <array identifier> = <mat formula>
<mat formula>::= <array identifier> | <mat function> | <array identifier> <mat operator>
    <array identifier> | <formula operand> * <array identifier>
<mat function>::= <mat initialization> | <mat initialization> <bound part> |
    INV <array parameter> | TRN <array parameter>
<mat initialization>::= ZER | CON | IDN (5)
<array parameter>::= <left bracket> <array identifier> <right bracket> (6)
<mat operator>::= + | - | * (7)

```

- (1) The <com statement>, if any exists, must be the first statement presented and have the lowest sequence number.
- (2) A sequence number may not exceed 9999 and must be non-zero.
- (3) A formal bound may not exceed 255 and must be non-zero.
- (4) Strings may not contain the quote character (").
- (5) A <bound part> for an IDN must be doubly subscripted.
- (6) An array may not be inverted or transposed into itself.
- (7) An array may not be replaced by itself multiplied by another array.

The Prepare BASIC System Program prepares a composite binary tape containing configured system input/output drivers, the significant locations in the system linkage area and, optionally, the BASIC compiler.

PBS is an absolute program containing unconfigured teletype, photo reader and punch drivers. The BASIC compiler will be included on the composite tape if it is loaded along with PBS by the Binary Loader.

When initiated, PBS requests I/O assignments for the teletype, photo reader and punch, configures the drivers in accordance with these assignments, and punches a binary tape containing the configured drivers and the BASIC compiler, if loaded. Multiple copies of this tape may be obtained.

OPERATING PROCEDURES FOR PBS

The system on which the composite tape is prepared must have a teletype ASR 33 or 35 in the same I/O channels as that of the system on which the compiler is to run. The tape is prepared in the following manner:

1. Load PBS using the Binary Loader.
2. Load the BASIC compiler, if desired.
3. Set the switch register to 000002. Press Load Address.
4. Set switches 5-0 of the switch register to the lower numbered channel assigned to the teletype.
5. Press Run.
6. PBS outputs the following request on the teletype:

PHOTO READER I/O ADDRESS?

Respond by inputting via the teletype keyboard the I/O channel number assigned to the photoreader followed by a carriage return (CR). If there is no photoreader, type a (CR) only.

7. PBS outputs the following request on the teletype:

PUNCH I/O ADDRESS?

Respond by inputting via the teletype keyboard the I/O channel number assigned to the punch followed by a (CR). If there is no punch, type a (CR) only.

8. Preparatory to punching the composite tape, PBS inquires if there is a high speed punch on the system, which is being used to prepare the tape, by outputting the request:

SYSTEM DUMP I/O ADDRESS?

- a) If there is a high speed punch on the system, input its I/O channel number on the keyboard following by a **CR**. PBS then proceeds to punch the composite tape on the high speed punch.

[Note if the preparation is performed on the computer on which BASIC is to run, the punch address given in 8 will be the same as that given in 7.]

- b) If there is no high speed punch on the system, input a **CR** only. PBS prints:

TURN ON TTY PUNCH, PRESS RUN

and halts the computer with 102011 showing on the T-register.

Turn on the TTY punch as instructed. When Run is pressed, PBS will proceed to punch the composite tape on the TTY punch.

9. When punching is complete, PBS will halt the computer with 102077 showing on the T-register. Further copies of the tape may be obtained by pressing Run after such a halt.

The following error conditions are detected by PBS:

1. If, in step 4, switches 5-0 of the switch register are set to a value less than 10_8 , PSB will halt the computer with 102055 showing on the T-register. To recover, set the switch register to the correct number and press Run.
2. If in steps 6, 7, 8 a number less than 10_8 or greater than 77_8 is input PSB will output the message:

INVALID I/O ADDRESS

To recover, input the correct number via the keyboard.

3. A channel number may consist only of two digits, no preceding or trailing blanks or other characters.

UNITED STATES

ALABAMA

P.O. Box 4207
2003 Byrd Spring Road S.W.
Huntsville 35802
Tel: (205) 881-4591
TWX: 810-726-2204

ARIZONA

3009 North Scottsdale Road
Scottsdale 85251
Tel: (602) 945-7601
TWX: 910-950-1282

232 South Tucson Boulevard
Tucson 85716
Tel: (602) 623-2564
TWX: 910-952-1162

CALIFORNIA

3939 Lankershim Boulevard
North Hollywood 91604
Tel: (213) 877-1282
TWX: 910-499-2170

1101 Embarcadero Road
Palo Alto 94303
Tel: (415) 327-6500
TWX: 910-373-1280

2591 Carlsbad Avenue
Sacramento 95821
Tel: (916) 482-1463
TWX: 910-367-2092

1055 Shafter Street
San Diego 92106
Tel: (714) 223-8103
TWX: 910-335-2000

COLORADO

7965 East Prentice
Englewood 80110
Tel: (303) 771-3455
TWX: 910-935-0705

CONNECTICUT

508 Tolland Street
East Hartford 06108
Tel: (203) 289-9394
TWX: 710-425-3416

111 East Avenue
Norwalk 06851
Tel: (203) 853-1251
TWX: 710-468-3750

DELAWARE

3941 Kennett Pike
Wilmington 19807
Tel: (302) 655-6161
TWX: 510-666-2214

FLORIDA

P.O. Box 545
Suite 106
9999 N.E. 2nd Avenue
Miami Shores 33153
Tel: (305) 758-3626
TWX: 810-848-7262

P.O. Box 20007
Herridon Station 32814
521 Commonwealth Avenue
Orlando
Tel: (305) 841-3970
TWX: 810-850-0113

P.O. Box 8128
Madera Beach 33708
1410 150th Avenue
St. Petersburg
Tel: (813) 391-0211
TWX: 810-863-0366

GEORGIA

P.O. Box 28234
450 Interstate North
Atlanta 30328
Tel: (404) 436-6181
TWX: 810-766-4890

ILLINOIS

5500 Howard Street
Skokie 60076
Tel: (312) 677-0400
TWX: 910-223-3613

INDIANA

4002 Meadows Drive
Indianapolis 46205
Tel: (317) 546-4891
TWX: 810-341-3263

LOUISIANA

P.O. Box 856
1942 Williams Boulevard
Kenner 70062
Tel: (504) 721-6201
TWX: 810-955-5524

MARYLAND

6707 Whitestone Road
Baltimore 21207
Tel: (301) 944-5400
TWX: 710-862-0850

MASSACHUSETTS

P.O. Box 1548
2 Choke Cherry Road
Rockville 20850
Tel: (301) 948-6370
TWX: 710-828-9684

MASSACHUSETTS

32 Hartwell Ave.
Lexington 02173
Tel: (301) 948-6370
TWX: 710-332-0382

MICHIGAN

24315 Northwestern Highway
Southfield 48075
Tel: (313) 353-9100
TWX: 810-232-1532

MINNESOTA

2459 University Avenue
St. Paul 55114
Tel: (612) 645-9461
TWX: 910-563-3734

MISSOURI

9208 Wyoming Place
Kansas City 64114
Tel: (816) 333-2445
TWX: 910-771-2087

2812 South Brentwood Blvd.
St. Louis 63144
Tel: (314) 644-0270
TWX: 910-760-1670

NEW JERSEY

W. 120 Century Road
Paramus 07652
Tel: (201) 265-5000
TWX: 710-990-4951

10508 N. Kings Highway
Cherry Hill 08034
Tel: (609) 667-4000
TWX: 710-852-4945

NEW MEXICO

P.O. Box 8366
Station C
6501 Lomas Boulevard N.E.
Albuquerque 87108
Tel: (505) 255-5586
TWX: 910-989-1665

156 Wyatt Drive
Las Cruces 88001
Tel: (505) 526-2485
TWX: 910-983-0550

NEW YORK

1702 Central Avenue
Albany 12205
Tel: (518) 869-8462
TWX: 710-441-8270

1219 Campville Road
Endicott 13764
Tel: (607) 754-0050
TWX: 510-252-0890

82 Washington Street
Poughkeepsie 12601
Tel: (914) 454-7330
TWX: 510-248-0012

39 Saginaw Drive
Rochester 14623
Tel: (716) 473-9500
TWX: 510-253-5981

1025 Northern Boulevard
Roslyn, Long Island 11576
Tel: (516) 869-8400
TWX: 510-223-0811

5858 East Mollay Road
Syracuse 13211
Tel: (315) 454-2486
TWX: 710-541-0482

NORTH CAROLINA

P.O. Box 5187
1923 North Main Street
High Point 27262
Tel: (919) 882-6873
TWX: 510-926-1516

OHIO

5579 Pearl Road
Cleveland 44129
Tel: (216) 884-9209
TWX: 810-421-8500

3460 South Dixie Drive
Dayton 45439
Tel: (513) 298-0351
TWX: 810-459-1925

OKLAHOMA

2919 United Founder Boulevard
Oklahoma City 73112
Tel: (405) 848-2801
TWX: 910-830-6862

OREGON

West Hills Mall, Suite 158
4478 S.W. Scholls Ferry Road
Portland 97225
Tel: (503) 252-9171
TWX: 910-464-6103

PENNSYLVANIA

2500 Moss Side Boulevard
Monroeville 15146
Tel: (412) 271-0724
TWX: 710-797-3650

144 Elizabeth Street
West Conshohocken 19428
Tel: (215) 248-1600, 828-6200
TWX: 510-660-8715

TEXAS

P.O. Box 1270
201 E. Arapaho Rd.
Richardson 75080
Tel: (214) 231-1601
TWX: 910-867-4723

P.O. Box 22813
4242 Richmond Avenue
Houston 77027
Tel: (713) 657-2407
TWX: 910-881-2645

GOVERNMENT CONTRACT OFFICE
225 Billy Mitchell Road
San Antonio 78226
Tel: (512) 434-4171
TWX: 910-871-1170

UTAH

2890 South Main Street
Salt Lake City 84115
Tel: (801) 486-8166
TWX: 910-925-5681

VIRGINIA

P.O. Box 6514
2111 Spencer Road
Richmond 23230
Tel: (703) 282-5451
TWX: 710-956-0157

WASHINGTON

433-108th N.E.
Bellevue 98004
Tel: (206) 454-3971
TWX: 910-443-2303

FOR U.S. AREAS NOT LISTED: Contact the regional office nearest you:
Atlanta, Georgia... North Hollywood, California... Paramus, New Jersey...
Skokie, Illinois. Their complete addresses are listed above.

CANADA

ALBERTA

Hewlett-Packard (Canada) Ltd.
11745 Jasper Ave.
Edmonton
Tel: (403) 482-5561
TWX: 610-831-2431

BRITISH COLUMBIA

Hewlett-Packard (Canada) Ltd.
304-1037 West Broadway
Vancouver 9
Tel: (604) 738-5301
TWX: 610-922-5059

MANITOBA

Hewlett-Packard Co. Ltd.
511 Bradford Ct.
St. James
Winnipeg
Tel: (204) 786-7581

NOVA SCOTIA

Hewlett-Packard (Canada) Ltd.
7001 Mumford Road
Suite 356
Halifax
Tel: (902) 455-0511
TWX: 610-271-4482

ONTARIO

Hewlett-Packard (Canada) Ltd.
880 Lady Ellen Place
Ottawa 3
Tel: (613) 722-4223
TWX: 610-562-1952

Hewlett-Packard (Canada) Ltd.

1415 Lawrence Avenue West
Toronto
Tel: (416) 249-9196
TWX: 610-492-2380

QUEBEC

Hewlett-Packard (Canada) Ltd.
275 Hymus Boulevard
Pointe Claire
Tel: (514) 697-4232
TWX: 610-422-3022
Tele: 01-20607

FOR CANADIAN AREAS NOT LISTED:

Contact Hewlett-Packard (Canada) Ltd. in Pointe Claire, at the complete address listed above.

CENTRAL AND SOUTH AMERICA

ARGENTINA

Hewlett-Packard Argentina
S.A.C.e.i
Lavalle 1171 - 3
Buenos Aires
Tel: 35-0436, 35-0627, 35-0431
Telex: 012-1009

BRAZIL

Hewlett-Packard Do Brasil
I.E.C. Ltda.
Rua Coronel Oscar Porto, 691
Sao Paulo - 8, SP
Tel: 71-1503
Cable: HEWPAK Sao Paulo
Hewlett-Packard Do Brasil
I.E.C. Ltda.
Avenida Franklin Roosevelt 84-
grupo 203
Rio de Janeiro, ZC-39, GB
Tel: 32-9733
Cable: HEWPAK Rio de Janeiro

CHILE

Hector Calcagni P.
Casilla 13942
Estado 215 - Oficina 1016
Santiago
Tel: 31-890, 490-505

COLOMBIA

Instrumentacion
Henrik A. Langebaek & Cia. Ltda.
Carrera 7 - 48-59
Apartado Aereo 6287
Bogota, 1 D.E.
Tel: 45-78-06, 45-55-46
Cable: AARIS Bogota

COSTA RICA

Lic. Alfredo Gallegos Gudián
Apartado Postal 3243
San José
Tel: 21-86-13
Cable: CALGUR San José

ECUADOR

Laboratorios de Radio-Ingeniería
Lab. Guayaquil 1246
Post Office Box 3199
Quito
Tel: 12496
Cable: HORVATH Quito

EL SALVADOR

Electrónica
Apartado Postal 1589
27 Avenida Norte 1133
San Salvador
Tel: 25-74-50
Cable: ELECTRONICA
San Salvador

GUATEMALA

Olander Associates Latin America
Apartado 1226
7a. Calle, 0-22, Zona 1
Guatemala City
Tel: 22812
Cable: OLALA Guatemala City

MEXICO

Hewlett-Packard Mexicana, S.A.
de C.V.
Apartado Postal 12-832
Eugenia 408, Dept. 1
Mexico 12, D.F.
Tel: 43-03-79, 36-08-78

NICARAGUA

Roberto Terán G.
Apartado Postal 689
Edificio Terán
Managua
Tel: 3451, 3452
Cable: ROTERAN Managua

PANAMA

Electrónica Balboa, S.A.
P.O. Box 4929
Ave. Manuel Espinosa No. 13-50
Bldg. Alina
Panama City
Tel: 30833
Cable: ELECTRON Panama City

PERU

Fernando Ezeta B.
Avenida Petit Thouars 4719
Miraflores
Casilla 3061
Lima
Tel: 50346
Cable: FEPERU Lima

PUERTO RICO

San Juan Electronics, Inc.
P.O. Box 5167
Ponce de Leon 154
Pda. 3 Pta. de Tierra
San Juan, P.R. 00906
Tel: (174) 725-3342
Cable: SATRONICS San Juan

URUGUAY

Pablo Ferrando S.A.
Comercial e Industrial
Avenida Italia 2877
Casilla de Correo 370
Montevideo
Tel: 40-3102
Cable: RADIUM Montevideo

VENEZUELA

Hewlett-Packard De Venezuela
C.A.
Edificio Arisán Office No. 6
Avda. Francisco de Miranda
Apartado del Este 10934
Chacaito
Caracas
Tel: 71.88.05, 71.88.69, 71.88.76
Cable: HEWPAK Caracas

FOR AREAS NOT LISTED,

CONTACT:
Hewlett-Packard Inter-Americas
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 325-7000
TWX: 910-373-1267
Cable: HEWPAK Palo Alto
Telex: 034-8461

EUROPE

AUSTRIA

Unilabor GmbH
Wissenschaftliche Instrumente
Rummelhardgasse 6/3
P.O. Box 33
Vienna 1X/71
Tel: 42 61 81
Cable: LABORINSTRUMENT
Vienna

BELGIUM

Hewlett-Packard Benelux S.A.
348 Boulevard du Souverain
Brussels 16
Tel: 72 22 40
Cable: PALOBEN Brussels
Telex: 23 494

DENMARK

Hewlett-Packard A/S
Egerang
Langebjerg 6
2850 Naerum
Tel: 01 80 40 40
Cable: HEWPAK AS
Telex: 66 40

FINLAND

Hewlett-Packard Oy
Gydenintie 3
Helsinki 20
Tel: 67 35 38
Cable: HEWPAK OY-Helsinki
Telex: 12-1563

FRANCE

Hewlett-Packard France
150 Boulevard Massena
75 Paris 13e
Tel: 707 97 19
Cable: HEWPAK Paris
Telex: 25048
Hewlett-Packard France
4 Quai des Etoiles
69 Lyon 5e
Tel: 42-63-45

GERMANY

Hewlett-Packard Vertriebs-GmbH
Lietzenburger Strasse 30
1 Berlin W 30
Tel: 24 86 36
Telex: 18 34 05

Hewlett-Packard Vertriebs-GmbH
Herrenberger Strasse 110
703 Böblingen, Württemberg
Tel: 07031-6671
Cable: HEPAG Böblingen
Telex: 72 65 739

Hewlett-Packard Vertriebs-GmbH
Achenbachstrasse 15
4 Düsseldorf 1
Tel: 68 52 56/59
Telex: 85 86 533

Hewlett-Packard Vertriebs-GmbH
Kurfürststrasse 95
6 Frankfurt 50
Tel: 52 00 36
Cable: HEWPAKSA Frankfurt
Telex: 41 32 49

Hewlett-Packard Vertriebs-GmbH
Beim Strohhause 26
2 Hamburg 1
Tel: 24 05 51/52
Cable: HEWPAKSA Hamburg
Telex: 21 53 32

Hewlett-Packard Vertriebs-GmbH
Reginfriedstrasse 13
8 München 9
Tel: 69 51 21/22
Cable: HEWPAKSA Munich
Telex: 52 49 85

GREECE

Kostos Karayannis
18, Ermou Street
Athens 126
Tel: 230 301
Cable: RAKAR Athens
Telex: 5962

IRELAND

Hewlett-Packard Ltd.
224 Bath Road
Slough, Bucks, England
Tel: Slough 33341
Cable: HEWPIE Slough
Telex: 84413

ITALY

Hewlett-Packard Italiana S.p.A.
Viale Lunginaga 46
20125 Milan
Tel: 69 15 84
Cable: HEWPAKIT Milan
Telex: 32046

Hewlett-Packard Italiana S.p.A.
Palazzo Italia
Piazza Marconi 25
00144 Rome - Eur
Tel: 591 2544
Cable: HEWPAKIT Rome
Telex: 61514

NETHERLANDS

Hewlett-Packard Benelux, N.V.
de Boelelaan 1043
Amsterdam, Z.2
Tel: 42 77 77
Cable: PALOBEN Amsterdam
Telex: 13 216

NORWAY

Hewlett-Packard Norge A/S
Nesvelen 13
Haslum
Tel: 53 83 60
Cable: HEWPAK Oslo
Telex: 6621

PORTUGAL

Telectra
Rua Rodrigo da Fonseca 103
P.O. Box 2531
Lisbon 1
Tel: 68 60 72
Cable: TELECTRA Lisbon
Telex: 1598

SPAIN

Atia Ingenieros
Urgel, 259
Barcelona, 11
Tel: 230-69-88
Atia Ingenieros
Enrique Larreta 12
Madrid, 16
Tel: 235 43 44
Cable: TELEATAID Madrid
Telex: 2 72 49

SWEDEN

HP Instrument AB
Hagakersgatan 7
Mölnådal
Tel: 031 - 27 68 00

HP Instrument AB

Svetsarvägen 7
Säna 1
Tel: 98 12 50
Cable: MEASUREMENTS
Stockholm
Telex: 10721

SWITZERLAND

HEWPAK AG
Zürcherstrasse 20
8952 Schlieren
Zürich
Tel: (051) 98 18 21
Cable: HEWPAKAG Zurich
Telex: 53933

HEWPAK A.G.

54 Route des Acacias
1211 Geneva 24
Tel: 43 79 29
Telex: 2 24 86

TURKEY

Telekom Engineering Bureau
P.O. Box 376 - Galata
Istanbul
Tel: 49 40 40
Cable: TELEATION Istanbul

UNITED KINGDOM

Hewlett-Packard Ltd.
224 Bath Road
Slough, Bucks
Tel: Slough 33341
Cable: HEWPIE Slough
Telex: 84413

YUGOSLAVIA

Belam S.A.
83 avenue des Mimosas
Brussels 15, Belgium
Tel: 34 33 32, 34 26 19
Cable: BELAMEL Brussels
Telex: 21790

FOR AREAS NOT LISTED,

CONTACT:
Hewlett-Packard S.A.
54 Route des Acacias
1211 Geneva, Switzerland
Tel: (022) 42 81 50
Cable: HEWPAKSA Geneva
Telex: 2.24.86

HEWLETT · PACKARD

AFRICA, ASIA, AUSTRALIA



SALES AND SERVICE

ANGOLA

Electra
Box 6487
Luanda
Cable: TELETRA Luanda

AUSTRALIA

Hewlett-Packard Australia
Pty. Ltd.
22-26 Weir Street
Glen Iris, 3146
Victoria
Tel: 20.1371 (4 lines)
Cable: HEWPARD Melbourne
Telex: 31024

Hewlett-Packard Australia
Pty. Ltd.
61 Alexander Street
Crows Nest 2065
New South Wales
Tel: 43.7866
Cable: HEWPARD Sydney

Hewlett-Packard Australia
Pty. Ltd.
97 Churchill Road
Prospect 5082
South Australia
Tel: 65.2366
Cable: HEWPARD Adelaide

CEYLON

United Electricals Ltd.
P.O. Box 681
Yahala Building
Staples Street
Colombo 2
Tel: 5496
Cable: HOTPOINT Colombo

CYPRUS

Kypricos
19-19D Hommer Avenue
P.O. Box 752
Nicosia
Tel: 6282-75628
Cable: HE-I-NAM

ETHIOPIA

African Salespower & Agency
Private Ltd., Co.
P.O. Box 718
58/59 Cunningham St.
Addis Ababa
Tel: 12285
Cable: ASACO Addisababa

HONG KONG

Schmidt & Co. (Hong Kong) Ltd.
P.O. Box 293
1511, Prince's Building
10, Charter Road
Hong Kong
Tel: 240168, 232735
Cable: SCHMIDTCO Hong Kong

INDIA

The Scientific Instrument
Co., Ltd.
8, Tej Bahadur Sapru Road
Allahabad 1
Tel: 2451
Cable: SICO Allahabad

The Scientific Instrument

Co., Ltd.
240, Dr. Dadabhai Naorji Road
Bombay 1
Tel: 26-2642
Cable: SICO Bombay

The Scientific Instrument

Co., Ltd.
11, Esplanade East
Calcutta 1
Tel: 23-4129
Cable: SICO Calcutta

The Scientific Instrument Co., Ltd.
30, Mount Road
Madras 2
Tel: 86339
Cable: SICO Madras

The Scientific Instrument Co., Ltd.
8-7, Ajmeri Gate Extn.
New Delhi 1
Tel: 27-1053
Cable: SICO New Delhi

IRAN

Telecom, Ltd.
P.O. Box 1812
240, Kh. Saba Shomali
Tehran
Tel: 43850, 48111
Cable: BASCOM Tehran

ISRAEL

Electronics & Engineering
Div. of Motorola Israel Ltd.
16, Kremenski Street
Tel-Aviv
Tel: 35021 (4 lines)
Cable: BASTEL Tel-Aviv
Telex: Bastel Tv 033-569

JAPAN

Yokogawa-Hewlett-Packard Ltd.
Nisei Ibaragi Bldg.
2-2-8 Kasuga
Ibaragi-Shi
Osaka
Tel: 0726-23-1641

Yokogawa-Hewlett-Packard Ltd.
Ito Building
No. 59, Kotori-cho
Nakamura-ku, Nagoya City
Tel: 551-0215

Yokogawa-Hewlett-Packard Ltd.
Ohashi Building
No. 59, I-chome, Yoyogi
Shibuya-ku, Tokyo
Tel: 370-2281
Telex: YHPMARKET TOK 23-724

KENYA

R. J. Tilbury Ltd.
P.O. Box 2754
Suite 517/518
Hotel Ambassadeur
Nairobi
Tel: 25670, 26803, 68206, 58196
Cable: ARJAYTEE Nairobi

KOREA

American Trading Co., Korea, Ltd.
P.O. Box 1103
Dae Kyung Bldg.
170 Sejong Ro
Chongro Ku
Seoul
Tel: 75-5841
Cable: AMTRACOE Seoul

LEBANON

Constantin E. Macridis
Clemenceau Street
Clemenceau Center
Beirut
Tel: 220846
Cable: ELECTRONUCLEAR Beirut

MALAYSIA

MECOMB Malaysia Ltd.
2 Lorong 13/5A
Section 13
Petaling Jaya, Selangor
Cable: MECOMB Kuala Lumpur

NEW ZEALAND

Hewlett-Packard (N.Z.) Ltd.
32.34 Kent Terrace
P.O. Box 9443
Wellington, N.Z.
Tel: 56-409
Cable: HEWPACK Wellington

PAKISTAN (EAST)

Mushko & Company, Ltd.
31, Jinnah Avenue
Dacca
Tel: 80058
Cable: NEWDEAL Dacca

PAKISTAN (WEST)

Mushko & Company, Ltd.
Osman Chambers
Victoria Road
Karachi 3
Tel: 51027, 52927
Cable: COOPERATOR Karachi

SINGAPORE

Mechanical and Combustion
Engineering Company Ltd.
9, Jalan Kilang
Singapore, 3
Tel: 642361-3
Cable: MECOMB Singapore

SOUTH AFRICA

F. H. Flanier & Co. (Pty.), Ltd.
Hill House
43 Somerset Rd.
Cape Town
Tel: 2-9711
Cable: AUTOPHONE Cape Town
Telex: 7038CT

F. H. Flanier & Co. (Pty.), Ltd.
607 Pharmacy House
80 Jorissen Street
Braamfontein, Johannesburg
Tel: 724-4172
Telex: 0026 JHB

TAIWAN

Hwa Sheng Electronic Co., Ltd.
P.O. Box 1558
21 Nanking West Road
Taipei
Tel: 46076, 45936, 48661
Cable: VICTRONIX Taipei

TANZANIA

R. J. Tilbury Ltd.
P.O. Box 2754
Suite 517/518
Hotel Ambassadeur
Nairobi
Tel: 25670, 26803, 68206, 58196
Cable: ARJAYTEE Nairobi

THAILAND

The International
Engineering Co., Ltd.
P.O. Box 39
614 Sukhumvit Road
Bangkok
Tel: 910722
Cable: GYSOM Bangkok

UGANDA

R. J. Tilbury Ltd.
P.O. Box 2754
Suite 517/518
Hotel Ambassadeur
Nairobi
Tel: 25670, 26803, 68206, 58196
Cable: ARJAYTEE Nairobi

VIETNAM

Landis Brothers and Company,
Inc.
P.O. Box H-3
216 Hien-Vuong
Saigon
Tel: 28.805
Cable: LANBROCOMP Saigon

ZAMBIA

R. J. Tilbury (Zambia) Ltd.
P.O. Box 2792
Lusaka
Zambia, Central Africa

FOR AREAS NOT LISTED,

CONTACT:
Hewlett-Packard Export
Marketing
3200 Hillview Ave.
Palo Alto, California 94304
Tel: (415) 326-7000
TWX: 910.373-1267
Cable: HEWPACK Palo Alto
Telex: 034-8461

HEWLETT-PACKARD

A Pocket Guide to Hewlett-Packard Computers

HEWLETT-PACKARD COMPANY 395 PAGE MILL ROAD PALO ALTO, CALIFORNIA 94301

